Observation-enhanced verification of operational processes.

Colin Paterson

University of York
Computer Science

March 2019

Abstract

Operational processes are at the core of many organisations. The failure and misuse of these processes can cause significant economic losses to businesses or, in the worst cases, endanger human life. As a result there has been significant research effort focused on the development of techniques and tools for the model-based analysis and verification of reliability, performance and quality-of-service properties of processes.

Constructing models which accurately represent the behaviour of real-world systems is very challenging. The complexity and stochastic nature of real-world phenomena requires the use of modelling assumptions which introduce errors that can significantly impact the results of model-based analysis. Where inaccurate analyses are used as the basis of engineering or business decisions, the consequences can be catastrophic.

Many operational processes are now routinely instrumented and capture information about component interactions and the behaviour of human operators. This thesis introduces a set of tool-supported techniques which exploit these logs in conjunction with tried and tested probabilistic model checking. This produces Markov models and formal analysis techniques which more accurately capture process behaviours and improve the quality of model-based analysis for operational processes.

We show how observation data can be used to improve the modelling and analysis of continuous time systems by refining continuous-time Markov models (CTMCs) to more accurately reflect real-world behaviours. We apply the tools and techniques developed to real-world processes and demonstrate how we may avoid the invalid decisions which arise from traditional CTMC modelling and analysis techniques.

We also show how observation-enhanced discrete time Markov models may be used to characterise the behaviour of users within an operational process. The self-adaptive role based access control approach we develop uses a formal definition of adaptation policies to identify potential threats in a real-world IT support system and mitigates risks to the system.

Contents

List of Figures

List of Tables

To Debbie.
I promise this is the last one.

Acknowledgements

I would like to acknowledge the time and effort that my supervisor, Dr. Radu Calinescu, has spent guiding my research over the last four years. Throughout my study he has been the perfect role model and I am sure that this work would have been much poorer without his input and patience.

I would like to thank the colleagues that I have been fortunate to work alongside during this project. I am especially grateful for the members of the Enterprise Systems group whose interesting and engaging discussions have meant there was never a dull moment. In particular I would like to thank Prof. Richard Paige, Prof. Fiona Polack, Dr. Yasmin Rafiq, Dr. Gabriel Costa Silva, Dr. Konstantinos Barmpis, Dr. Ran Wei and Dr. Athanasios Zolotas.

Special thanks go to Dr. Simos Gerasimou who was always available to listen when I needed to talk through challenging problems.

I would like to thank DSTL for their support in making this project a reality. Their input throughout has provided valuable insights which have helped to shape the work.

During my study I have been lucky to collaborate with some great researchers many of whom have become good friends. A special thank you goes to Dr. Kenneth Johnson, Dr. Carlos Eduardo Da Silva and José Diego Saraiva da Silva.

Finally I would like to thank my family who, despite doubting my sanity in starting this PhD, have always supported me. It may have been my daft idea to start with but it's their love and support that has allowed me to complete this new adventure.

Declaration

I declare that this thesis is a presentation of original work and I am the sole author. This work has not previously been presented for an award at this, or any other, University. All sources are acknowledged as References.

Chapter 6 describes contributions that involved collaborative worth with with Dr. Kenneth Johnson of the Auckland University of Technology and Chapter 8 describes contributions that involved collaborative work with Dr. Carlos Eduardo daSilva and José Diego Saraiva da Silva of the Federal University of Rio Grade do Norte, Brazil. The parts where this is the case are clearly identified in Section 1.2.

Parts of the research described in this thesis have previously published in:

- Radu Calinescu, Kenneth Johnson and Colin Paterson (2015). **FACT: A Probabilistic Model Checker for Formal Verification with Confidence Intervals**. International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS), 2016.
- Carlos Eduardo da Silva, Jose Diego Saraiva da Silva, Colin Paterson and Radu Calinescu (2017). **Self-Adaptive Role-Based Access Control for Business Processes** Proceedings of the 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS). IEEE Press, 2017.
- Colin Paterson and Radu Calinescu (2017). **Accurate Analysis of Quality Properties of Software with Observation-Based Markov Chain Refinement**. IEEE International Conference on Software Architecture (ICSA), 2017
- Colin Paterson and Radu Calinescu (2018). **Observation-Enhanced QoS Analysis of Component-Based Systems**. IEEE Transactions on Software Engineering, 2018.
- Radu Calinescu, Colin Paterson and Kenneth Johnson (2018): **Efficient parametric model checking using domain-specific modelling patterns**. Proceedings of the 40th International Conference on Software Engineering: New Ideas and Emerging Results, 2018.

Part I

Introduction and Background

Chapter 1

Introduction

1.1 Motivation

Complex systems are now a part of everyday life with software and hardware components integrated to create business and safety-critical applications. These applications form the core of many operational processes, the failure of which may cause significant economic loss to businesses and may even endanger human life. Ensuring the correctness of such systems has become a significant area of research with system monitoring, modelling and analysis utilised to validate and verify the compliance of systems against functional and Quality-of-Service (QoS) requirements.

Model checking has been suggested as one way to ensure the correctness of systems, providing organisations with a level of assurance that the system will operate as expected [6]. Model checking provides mathematical guarantees that system requirements are met and, where requirements are violated, counter examples are provided as proof of failure. For hardware and software design, model checking has become a commonplace activity. Whilst conventional model checking focuses on an absolute guarantee of correctness with respect to the system models, e.g. "the system will never fail", this may not be appropriate for real-world systems and processes in which stochastic variations exist. In such cases a probabilistic view of the system allows for model-based performance evaluation [7] e.g. "The probability of failure for the system within one year is below 0.01". The verification of probabilistic models can be undertaken through Monte Carlo based statistical techniques which provide an estimation of results with bounds. The results provided by these techniques will eventually converge to a correct answer but may require a very large number of simulation runs to do so. Typically such techniques provide a confidence bound on the result and therefore limited guarantees on the accuracy of the analysis. By contrast probabilistic model checking tools [8, 9, 10, 11] have been developed which allow for the verification of such models and provide mathematically provable verification results. These tools have matured over the past decade and found use in a wide range of contexts [12, 13].

One area where probabilistic model checking offers potential benefits is in the verification of operational processes such as, air traffic control [14][15], pharmaceutical dispensing [16] and e-Government service delivery [17]. Within this work we define a process as "a specific

ordering of work activities across time and place, with a beginning, an end, and clearly identified inputs and outputs" [18]. An operational process is then those processes which enact the operations function of an organisation where the operations function delivers those products and services which are the reason for existence of the organisation [19]. These operational processes are critical to the success of modern organisations, and at the heart of such processes are socio-technical systems which combine electronic components and human participants to fulfil operational requirements.

The performance, cost, resource use and other quality-of-service (QoS) properties of these systems and processes underpin important engineering and business decisions and failures may lead to financial loss or, in the worst cases, the loss of life.

Formal verification is dependent on the construction of mathematical models which accurately represent the system of interest. Modelling real-world phenomena is often difficult or even impossible, however, and where modelling assumptions are incorrect the guarantees provided by formal verification are invalid. Uncertainty in models may arise from two primary sources. Firstly a lack of knowledge concerning the behaviour of system components and their interactions and; secondly from the dynamic and unpredictable behaviour of the environments in which the process operates. The challenge of verifying models which incorporate uncertainty is particularly relevant for operational processes involving human participants. In such models the value of model parameters which characterise the probability of actions occurring or the temporal characteristics of operations are uncertain and may be obtained through limited observations or domain knowledge. In addition human centric processes are typically constructed in the knowledge that flexibility exists within the process. Flexibility allows processes to adapt to changing environments and comes in a number of forms [21]:

- Flexibility by design: to allow for a range of possible actions depending on the context of the process instance e.g. high and low priority cases require different authorisation activities,
- Flexibility by deviation to allow for a process instance to deviate at run-time without the need to alter the published process e.g. we treat a patient before registering them as the computer system is currently unavailable.
- Flexibility by underspecification: only after deployment will it become clear what the precise steps required to complete a task are, and hence an abstract state is created with the aim of future refinement.

Allowing flexibility within processes may allow organisations to gain a competitive advantage by responding to environmental change [21], and a process that does not support change will not benefit the organisation in the long run [22].

Whilst generating models for operational processes is difficult the need for accuracy is paramount since a process model which does not sufficiently capture the behaviour of the real world is of little value and any verification based on that model may lead to invalid business decisions.

In the area of probabilistic model verification, modelling typically relies on domain expertise to estimate model parameters such as state transition probabilities and rates. These are

insufficient to capture many real-world behaviours. As operational processes have become more complex, the amount of instrumentation applied to these processes has also increased with logs of activity routinely stored for future analysis. This provides an opportunity to develop modelling and analysis techniques which more accurately represent observed behaviours and better support decision making. It is this opportunity which we exploit in this work.

1.2 Contributions

The primary hypothesis of this thesis is that by combining observation data with traditional modelling techniques we may more accurately represent the behaviour of operational processes using Markovian models. When probabilistic model checking is applied to the resulting models the results generated will better support business and engineering decisions than traditional techniques.

In this PhD project we have developed a set of tool-supported techniques which allow for observation data to be used in the modelling and analysis of continuous and discrete time systems. The main contributions of the thesis are summarised below.

1. **Improving the accuracy of CTMC model verification using observation data.** Chapter 3 introduces OMNI, an observation-based Markov chain refinement approach. OMNI tackles the challenge of generating mathematical models which are sufficiently accurate to support the design and verification of real-world systems and processes. We demonstrate how phase-type distributions can be used to significantly improve the accuracy of continuous time Markov chains models and hence improve the quality of decisions based on these models. Using observation data obtained from real-world web-services, from which we construct a web-based travel application, the OMNI refinement method reduces QoS analysis errors by between 77–90.3% thus reducing the risk of invalid decisions.

2. **Controlling the size of CTMC models using property-centric model refinement.** Chapter 4 extends OMNI and introduces an approach for the classification of Markov model states with reference to formal verification properties. This approach allows for the generation of significantly smaller Markov chains whilst retaining the accuracy gains achieved using observation-based model enhancement. We demonstrate our approach on the web-based travel application from Chapter 3 as well as a second real-world IT support system and show how verification times are reduced by 54–74%.

3. **OMNI.** In Chapter 5 we present the OMNI tool which implements observation-enhanced QoS analysis of component based systems using automatic component classification and integrates with standard probabilistic model checkers. We have made this tool freely available together with the two case studies used to evaluate the tool as well as the models and datasets required to replicate our work. `https://www.cs.york.ac.uk/tasp/OMNI/index.html`

4. **FACT: A tool for the formal verification of DTMCs with confidence intervals.** In Chapter 6 we present the FACT probabilistic model checking tool which

computes confidence intervals for the evaluation of DTMCs with unknown transition probabilities when observations of these transitions are available. FACT is joint work with Kenneth Johnson from Auckland University of Technology, New Zealand and builds on theoretical foundations developed by Radu Calinescu et al. [23]. FACT integrates existing components developed by Kenneth Johnson with a graphical user interface to allow users to more easily leverage the underlying theory. We evaluate FACT using a set of case studies across a range of application domains and the resulting tool and case studies are freely available on our website `https://www-users.cs.york.ac.uk/~cap/FACT/`.

5. **Detecting abnormal behaviour in business processes using observation data.** We present a formal method for the definition of adaptation policies in operational processes. User behaviours are analysed in line with the policies to assess normality and to mitigate potential threats. The approach integrates the FACT model checker and parametric model templates to characterise user behaviour with respect to policy requirements and to determine if user access permissions should be amended.

6. **Evaluation of abnormal behaviour detection applied to a real-world application.** In Chapter 8 we present an evaluation of our approach with reference to the real-world IT-Support system introduced in Chapter 4. We demonstrate how the approach is able to capture a range of real-world policies and limit the potential for negative behaviours to impact system performance. The instrumentation of the process, the implementation of the analysis approach and discussions with the IT management staff was undertaken by our collaborators, Carlos Eduardo da Silva and José Diego Saraiva da Silva at the IFRN[1] in Brazil.

7. **Case studies.** Throughout this work we present two case studies and the data sets which were captured from these real-world systems. The first case study is a web services orchestration and demonstrates the challenges of working with components which possess deterministic delays and whose timing characteristics are non-parametric. The second case study presents a human centric operational process and as such has highly variable timing characteristics. These two case studies together provide a opportunity for other researchers to understand the challenges of applying probabilistic verification to real-world systems and processes.

1.3 Thesis structure

The work presented in the thesis is organised into four parts as shown in Figure 1.1. Part I includes the introduction and background information which underpins this work. Part II then consists of three chapters which concern the analysis of continuous time systems and the benefits which can be achieved when observation data exists. Part III switches to discrete time systems and examines how observation data may be used to enhance the analysis of system behaviours. The thesis concludes in Part IV with two chapters which summarises findings of the work and suggesting areas of future work.

Chapter 2 introduces the key concepts, mathematical models and techniques that underpin

[1] Federal Institute of Education, Science and Technology of Rio Grande do Norte

<table>
<tr><td colspan="2">Part I: Introduction (Ch. 1) & Background (Ch. 2)</td></tr>
<tr><td>Part II : Continuous Time

Improving the accuracy of CTMC model verification using observation data. [3] (Ch. 3)

Controlling the size of CTMC models using property-centric model refinement. [4] (Ch. 4)

Evaluation of observation based refinement approaches. [4] (Ch. 5)</td><td>Part III : Discrete Time

FACT: A tool for the formal verification of DTMCs with confidence intervals. [1] (Ch. 6)

Detecting abnormal behaviour in business processes using observation data. [2] (Ch. 7)

Evaluation of abnormal behaviour detection applied to a real-world application. [2] (Ch. 8)</td></tr>
<tr><td colspan="2">Part IV: Conclusions (Ch. 9) & Future Work (Ch. 10)</td></tr>
</table>

Figure 1.1: Thesis structure - References refer to the papers in which contributions were first presented

the research carried out by the PhD project. This includes continuous and discrete time Markov models and the temporal logics that are used to specify properties for those models. In addition phase-type distributions are introduced since these are key to understanding the Markov chain refinement work undertaken. Finally we discuss probabilistic model checking software which is used extensively throughout the rest of the thesis.

Chapter 3 presents OMNI, an approach for improving the accuracy of continuous time Markov chain (CTMC) model verification using observation data. In Section 3.1 we motivate this work by considering a web based travel application composed of real-world web services and demonstrate how traditional analysis using CTMC models may lead to invalid engineering decisions. Section 3.2 describes the OMNI approach and shows how, by utilising log data gathered from system components, we are able to refine component models and hence increase the accuracy of the model analysis. The two stage OMNI process uses Erlang distribution modelling of deterministic delays (Section 3.2.2) and phase-type distributions to model residual holding time (Section 3.2.3). Finally Section 3.3 compares our approach to related research.

Chapter 4 then extends OMNI introducing a property-centric refinement approach which reduces the size of the models produced whilst maintaining the accuracy of the observation based refinement approach. Section 4.1 describes the activity classification step of the extended OMNI approach. This step partitions the states of the high-level CTMC into subsets that require different amounts of refinement because of the different impact of their associated activities on the analysed property. The second step, selective refinement, is described in Section 4.2 and shows how the methods developed in Chapter 3 are applied to the newly classified stated. The chapter closes with a discussion of related work in Section 4.3.

In Chapter 5 we evaluate the property-centric OMNI approach developed in the previous chapters. Section 5.1 describes the OMNI tool which allows for the definition of observation-

enhanced CTMC models and the language used to specify these models. In Section 5.2 we then evaluate the OMNI approach with reference to the motivating example introduced in Chapter 3 and a human centric IT support system currently implemented at a university in Brazil. Finally, threats to the validity of our evaluation are considered in Section 5.3.

Chapter 6 presents FACT, a tool for the formal analysis of discrete time Markov chain models using confidence intervals. In Section 6.1 we describe the theoretical foundations developed by Calinescu et al. [23] upon which FACT is built. Section 6.2 then describes the FACT tool, its architecture and the extension to the PRISM language which allows for models to be defined with observation counts. In Section 6.3 we show how FACT may be used to analyse a web application before evaluating FACT using five benchmark case studies in Section 6.4. The chapter concludes with a discussion of related work in Section 6.5.

Chapter 7 presents a self adaptive role based access control framework (saRBAC) which allows for the identification of abnormal behiour in operational processes through the analysis of trace log data. Section 7.1 presents our motivating example based on the IT support system first introduced in Chapter 5 as well as a set of informal policies specified by the management team in natural language. Section 7.2 then describes the five stage saRBAC approach and presents a method for formalising policies using Probabilistic computation tree logic (PCTL) and first-order logic. Examples in this section utilise the FACT analysis engine introduced in the previous chapter. Finally Section 7.3 discusses related work.

Chapter 8 presents an evaluation of the saRBAC approach using data logs gathered from the IT support system over a period of 60 days. Section 8.1 presents the architecture of the saRBAC implementation and describes how the method interfaces with the operational process and utilises the FACT model checker for model analysis. Section 8.2.1 then presents an evaluation of the approach in three parts. Firstly the policies are calibrated using real data and we demonstrate that the policies are sufficient to recognise abnormal behaviours in the trace data. Next we validate the policies using a set of traces which were not used in the calibration process. In Section 8.2.3 we assess the performance of the implementation before finally considering threats to the validity of our evaluation, in Section 8.3.

In the final part of the thesis Chapter 9 summaries the insights gained from the PhD project and the contributions presented before Chapter 10 concludes the thesis with a range of areas for further research which can extend or build on the results of the PhD project.

Chapter 2

Background

System verification is undertaken to ensure correctness and to support quality control against a specification for systems design. In practice much of this work is carried out by hand using peer reviewed dynamic testing. Such techniques require the development of prototypes and tests which can prove the existence of errors rather than guaranteeing their absence. Furthermore, as the complexity of systems increases such approaches become less effective in finding errors.

Formal verification, however, does not require a completed prototype, but instead utilises a formal specification and derived models for the system under investigation. In this way formal verification can be considered a mathematical proof of correctness with respect to model correctness.

In this chapter we present modelling and analysis techniques which allow for the formal verification of system models. These underpin the work undertaken throughout the thesis. Section 2.1 defines the concept of model checking and its role in formal verification. Section 2.2 then defines discrete-time and continuous-time Markov models and the formal temporal logics which may be used to analyse system properties of models defined using each time variant. In Section 2.3 we define phase-type distributions (PHDs) which utilise continuous Markov chains before introducing two forms of PHD, Erlang and Hyper-Erlang distributions, which are used in this work. Finally Section 2.4 briefly outlines the use of probabilistic model checking software.

2.1 Model checking

Model checking originates in the independent work of two research teams Clarke and Emerson [24], and Quielle and Sifakis [25]. In 2007, Clarke, Emerson and Sifakis won the Turing Award for 25 years of work which has defined the field [6]. In that time model checking has become a well accepted technique for the verification of hardware and software systems providing

> ... an algorithmic means of determining whether an abstract model represent-

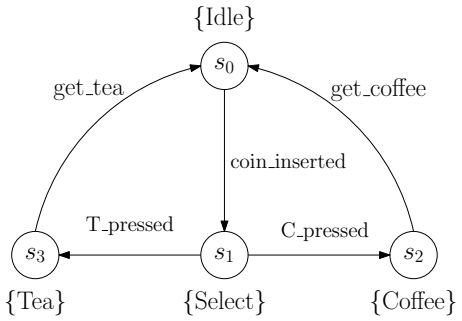


Figure 2.1: State transition system of a simple vending machine

ing, for example, a hardware or software design satisfies a formal specification expressed as a temporal logic formula [6].

In model checking we wish to verify that a formal model of the system under review, M, satisfies a set of system properties, ϕ, specified in a precise language, such that

$$M \vDash \phi. \tag{2.1}$$

System models are typically abstracted from the system under consideration as state transition systems. Figure 2.1 shows a simple four state model of a vending machine, adapted from [26]. Starting in state s_0 the system moves to s_1 when a coin is inserted. The system moves to either s_2 or s_3 contingent on the action of the user before dispensing the appropriate drink and returning to the idle state.

From such a representation we can then examine properties of the system to ensure correct operation, where properties are defined using formal logics.

Whilst model checking focuses on absolute guarantees of correctness for example "the system will never reach deadlock", for many real-world problems stochastic behaviours exist and, as such, rigid guarantees are not possible. Under such conditions we turn to probabilistic model checking techniques, where models are extended to include probabilities.

2.2 Markov models

Markov models are transition systems in which choice in the system is modelled as a probability. Certain behaviours of the system may then be modelled as stochastic processes, where a *stochastic process* is defined as a family of random variables

$$\{X(t),\ t\ \in\ T\}.$$

$X(t)$ is a random variable and as such is defined on a probability space. T is the index set or parameter space and t is normally assumed to be time such that $X(t)$ denotes the value of the random variable at time t. For a *discrete-time* stochastic process $T = \{0, 1, 2, \cdots\}$ whilst for continuous-time processes $T = \{t : 0 \leq t < \infty\}$. The values assumed by $X(t)$

are the system states and the set of all possible states is known as the state space. When $X(t) = s$ the process is said to be in state s at time t. Where the state space is discrete the process is referred to as a chain and the states can be identified as a set of natural numbers.

A stochastic process is said to be Markovian if the probability of the next transition is dependent only on the current state and not on any previous states i.e. the system is memoryless.

Definition 2.1. *A stochastic process* $\{X(t)|t = 0, 1, 2, \cdots\}$ *satisfies the Markov property if*

$$P\{X_{t+1} = s_{t+1}|X_t = s_t, X_{t-1} = s_{t-1}, \cdots X_1 = s_1, X_0 = s_0\} = P\{X_{t+1} = s_{t+1}|X_t = s_t\}$$

where $s_0, s_1, \cdots s_k$ *represent successive states of the process.*

2.2.1 Discrete-time Markov chains

A discrete-time Markov chain is a state-transition system in which the state is observed at discrete times. The system is assumed to only occupy a single state at any point in time and transitions between states are assumed to be instantaneous. The next state at each point in time is specified by a probability distribution.

2.2.1.1 Describing a state transition system as a DTMC

The following definition of a Discrete Time Markov Chain (DTMC) is adapted from [27]:

Definition 2.2. *A labelled Discrete Time Markov Chain (DTMC) is a tuple:* $\mathcal{D} = (S, \boldsymbol{P}, \boldsymbol{\pi}_0, AP, L)$ *where*

- S *is a countable, nonempty set of states;*
- $\boldsymbol{P} : S \times S \to [0, 1]$ *is the transition probability matrix such that for all states* s*:*

$$\sum_{s' \in S} \boldsymbol{P}(s, s') = 1; \tag{2.2}$$

- $\boldsymbol{\pi}_0$ *is the initial distribution such that* $\sum_{s \in S} \boldsymbol{\pi}_0(s) = 1$ *and*
- AP *is a set of atomic propositions and* $L : S \to 2^{AP}$ *is a labelling function.*

Time in the DTMC is abstract and movement between states may be considered to occur at discrete moments in time. This may be clock ticks of a hardware system or an arbitrary unit of time decided by the modeller.

The transient probability distribution $\boldsymbol{\pi}_n$ for a DTMC may be calculated at time step t_n from an initial state $\boldsymbol{\pi}_0$ as:

$$\boldsymbol{\pi}_n = \boldsymbol{\pi}_0 \times \boldsymbol{P}^n. \tag{2.3}$$

The labelling of states allows for the allocation of meaningful names to states and allows us

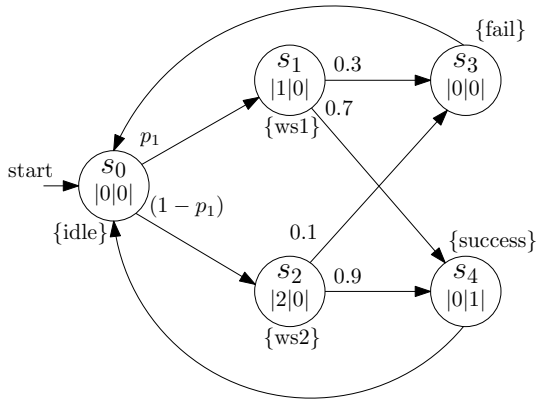


Figure 2.2: DTMC annotated with state rewards and costs

to ask questions such as "what is the probability of running for 10 steps without reaching the fail state".

2.2.1.2 Extending DTMCs with rewards

Quantitative analysis of Markov models may be enhanced through the addition of positive real valued quantities associated with states and transitions in the model. These values may then be interpreted as either rewards or costs which the analyst will aim to maximise or minimise respectively[1].

Definition 2.3. *Given a DTMC $\mathcal{D} = (S, \boldsymbol{P}, \boldsymbol{\pi}_0, AP, L)$, a reward structure of $\mathcal{D}$ is a pair of real-valued functions $(\underline{\rho}, \iota)$ where:*

- $\underline{\rho} : S \rightarrow \mathbb{R}_{\geq 0}$ *: is a state reward function that defines the value (reward/cost) associated with being in state $s \in S$ for one time step.*
- $\iota : S \times S \rightarrow \mathbb{R}_{\geq 0}$ *: is a transition reward function that defines the value associated with each transition in the model.*

Example 2.1. *Figure 2.2 illustrates a software system which calls one of two web services, ws1 (s_1) or ws2 (s_2). ws1 is cheap to call but has a failure probability of 0.3 whilst ws2 is more expensive but has a lower failure rate, 0.1. The failure rates are reflected in the transition probabilities from each web service to the failure state (s_3).*

[1] Whilst we refer to costs and rewards the difference is purely semantic. There is no difference, mathematically, in how the quantities are evaluated.

The DTMC is then:

$$
\begin{aligned}
S &= \{s_0, s_1, s_2, s_3, s_4\} \\
\boldsymbol{\pi}_0 &= [1, 0, 0, 0, 0] \\
AP &= \{idle, ws1, ws2, fail, success\} \\
L(s_0) &= \{idle\} \\
L(s_1) &= \{ws1\} \\
L(s_2) &= \{ws2\} \\
L(s_3) &= \{success\} \\
L(s_4) &= \{fail\}
\end{aligned}
\qquad
\boldsymbol{P} = \begin{pmatrix}
0 & p_1 & (1-p_1) & 0 & 0 \\
0 & 0 & 0 & 0.3 & 0.7 \\
0 & 0 & 0 & 0.1 & 0.9 \\
1 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0
\end{pmatrix}
$$

A systems designer wishes to maximise the number of successful calls whilst minimising the cost associated with service calls and will select p_1 (the probability of calling ws_1) appropriately.

To achieve this, the model is augmented with costs and rewards which represent the monetary costs associated with calling the web service and reward for successfully completing a call. In this case the designer chooses to place costs and rewards on states leaving all transitions with zero rewards. These are annotated below the state and labelled as :

$$| < \textit{cost of calling service} > | < \textit{reward for completion} > |$$

such that s_4 has a reward of 1, denoting a successful service call, and s_1 and s_2 have costs of 1 and 2 respectively reflecting the price differential between them. The reward structures are therefore defined as

$$
\begin{aligned}
\iota &= 0_{5,5} \\
\underline{\rho}^{cost} &= (0, 1, 2, 0, 0) \\
\underline{\rho}^{reward} &= (0, 0, 0, 0, 1)
\end{aligned}
$$

where $0_{5,5}$ denotes a 5×5 matrix with the value zero in every element.

2.2.1.3 Analysing properties of a DTMC

Through model checking we wish to employ tools and techniques which allow for the formal verification of system properties specified in a precise language such that our model $\mathcal{D}$ satisfies a specification ϕ.

$$\mathcal{D} \vDash \phi \tag{2.4}$$

Probabilistic computation tree logic (PCTL) [28] is a branching time temporal logic (based on CTL [29]) which allows for the specification of probabilistic properties of a DTMC and hence for the formal definition of specifications.

Properties are defined with reference to states and possible execution paths within the model

$\mathcal{D}$ starting from an initial state s_0. An execution path ω is a non-empty sequence of states $s_0, s_1, \cdots$ where $s_i \in S$ and $P(s_i, s_{i+1}) > 0$ for all $i \geq 0$. Paths may be finite or infinite and we denote the i^{th} state of a path as ω_i. The set of all paths starting in state s is denoted $Path^{\mathcal{D}}(s)$.

Model behaviour is then analysed by assessing the probability that a certain path is taken and this is calculated for each state $s \in S$ by defining a probability measure Pr_s over the set of paths $Path^{\mathcal{D}}(s)$ [27].

Definition 2.4. *The syntax of PCTL is defined as follows:*

$$\begin{aligned} \Phi &::= true \mid a \mid \neg\Phi \mid \Phi \wedge \Phi \mid P_{\bowtie p}[\phi] \\ \phi &::= X\ \Phi \mid \Phi\ U^{\leq k}\ \Phi \end{aligned}$$

and the cost/reward augmented PCTL state formulae are defined as:

$$R_{\bowtie r}[C^{\leq k}] \mid R_{\bowtie r}[I^{=k}] \mid R_{\bowtie r}[F\ \Phi]$$

where $a \in AP$, $\bowtie\ \in \{<, \leq, \geq, >\}$ is a relational operator, $p \in [0,1]$ is a probability bound, $k \in \mathbb{N} \cup \{\infty\}$ and $r \in R_{\geq 0}$ is a reward bound. Φ and ϕ denote state and path formulae respectively.

The path formula ϕ imposes a condition on the set of paths from a system such that $P_{\bowtie p}[\phi]$ is the probability that the set of paths which satisfy ϕ meet the bounds defined by $\bowtie p$. For a path ω, the next operator $X\ \Phi$ holds if Φ is satisfied in the next state. The "bounded until" operator, $\Phi_1\ U^{\leq k}\Phi_2$, holds if Φ_2 becomes true within k time steps and Φ_1 is true at all time steps before this.

The reward operators may be interpreted as follows:

- $R_{\bowtie r}[C^{\leq k}]$: The amount of reward accumulated along a path ω up to time step k meets the bound defined by $\bowtie r$;
- $R_{\bowtie r}[I^{=k}]$: The expected reward at time step k satisfies the bound defined by $\bowtie r$;
- $R_{\bowtie r}[F\ \Phi]$: The expected reward accumulated before reaching a state which satisfies Φ satisfies $\bowtie r$. Note that $F\ \Phi$ may be alternatively written as $\mathsf{true}\ U^{\leq\infty}\ \Phi$.

The semantics of PCTL are defined with a satisfaction relation $\vDash$ over the states, $s \in S$ and paths $\omega \in Path^{\mathcal{D}}$ of a DTMC by:

$$\begin{array}{ll} s \vDash true & \text{for all } s \in S \\ s \vDash a & \text{iff } \ a \in L(s) \\ s \vDash \neg\Phi & \text{iff } \ s \nvDash \Phi \\ s \vDash \Phi_1 \wedge \Phi_2 & \text{iff } \ s \vDash \Phi_1 \text{ and } s \vDash \Phi_2 \\ s \vDash \mathbb{P}_{\bowtie p} & \text{iff } \ Prob^{\mathcal{D}}(s, \phi) \bowtie p \end{array}$$

where

$$Prob^{\mathcal{D}}(s, \phi) = Pr_s(\omega \in Paths^{D}(s) \mid \omega \vDash \phi)$$

Table 2.1: Examples of PCTL formulae

Description	PCTL formula
The probability of of reaching the *fail* state within 100 time units is at less than 95%	$P_{<0.95}$ [**true** $U^{\leq 100}$ $fail$]
The probability of reaching the *success* state without first visiting the *fail* state is at least 90%	$P_{\geq 0.90}$ [$\neg$ $fail$ U $success$]
The expected cost to reach the *success* state will be at most 5	$R^{cost}_{\leq 5}$ [F $success$]

and for any path $\omega \in Path^{\mathcal{D}}(s)$:

$$\begin{array}{lll} \omega \vDash & X\Phi & \text{iff} \quad \omega(1) \vDash \Phi \\ \omega \vDash & \Phi_1 U^{\leq k} \Phi_2 & \text{iff} \quad \exists i \in \mathbb{N}.(i \leq k \wedge \omega(i) \vDash \Phi_2 \wedge \forall\, j < i.(\omega(j) \vDash \Phi_1)) \end{array}$$

For a DTMC extended with rewards the semantics are defined as follows:

$$\begin{array}{rll} s \vDash R_{\bowtie r}[C^{\leq k}] & \text{iff} & Exp^{\mathcal{D}}(s, Y_{C \leq k}) \bowtie r \\ s \vDash R_{\bowtie r}[I^{=k}] & \text{iff} & Exp^{\mathcal{D}}(s, Y_{I=k}) \bowtie r \\ s \vDash R_{\bowtie r}[F\Phi] & \text{iff} & Exp^{\mathcal{D}}(s, Y_{F\Phi}) \bowtie r \end{array}$$

where $Exp^{\mathcal{D}}(s, Y)$ denotes the expectation of the random variable $Y : Paths^{\mathcal{D}}(s) \to \mathbb{R}_{\geq 0}$ with respect to the probability measure Pr_s and for any path ω [27].

Example 2.2. *The software system described in Example 2.1 may be analysed using PCTL in order to assess the proposed system against a requirements specification. The requirements and PCTL formulae which encode them are provided in Table 2.1.*

2.2.2 Continuous-time Markov chains

The continuous time Markov chain (CTMC) is a type of stochastic state transition model used for Quality of Service (QoS) analysis at both design time [30],[31],[32] and runtime [33],[34]. CTMCs provides a representation for discrete state, continuous time models and may be considered as a DTMC where transition times occur according to a Poisson process [35]. In such models the transition between states does not occur at discrete-time steps but at any point in time. The CTMC, like the DTMC, possesses the Markov property and is therefore memoryless such that the next state is only dependent on the current state and not previously visited states. In addition the amount of time spent in previous states is also irrelevant.

2.2.2.1 Describing a state transition system as a CTMC

Definition 2.5. *A CTMC over an atomic proposition set AP is a tuple:* $\mathcal{C} = (S, \boldsymbol{R}, \boldsymbol{\pi}_0, AP, L)$ *where*

- S *is a countable, nonempty set of states;*
- $\boldsymbol{R} : S \times S \to \mathbb{R}_{\geq 0}$ *is the transition rate matrix*

- $\boldsymbol{\pi}_0$ *is the initial distribution such that* $\sum_{s \in S} \boldsymbol{\pi}_0(s) = 1$*;*
- AP *is a set of atomic propositions and* $L : S \to 2^{AP}$ *is a labelling function.*

The transition rate matrix R describes the rate with which the system will transition between states where rates are assumed to have an exponential distribution. The exponential distribution is chosen as it is the only continuous distribution that exhibits the memoryless property i.e. the expected time for which a system will stay in state s is independent of the time already spent in the state. Using this assumption, transitions between two states, i and j can happen if $\boldsymbol{R}(s_i, s_j) > 0$ and the probability of a transition within t time units is $(1 - e^{-t\boldsymbol{R}(s_i, s_j)})$.

The time spent in any state s, before a transition occurs is then exponentially distributed with a rate:

$$E(s) = \sum_{s' \in S} \boldsymbol{R}(s, s') \tag{2.5}$$

where $E(s)$ is known as the exit rate of s. A state s_i is said to be absorbing if $\boldsymbol{R}(s_i, s_j) = 0$ for all $s_j \in S \backslash \{s_i\}$ and transient otherwise.

We can also determine the probability of being in state s' next independently of the time spent in s by extracting the embedded DTMC from the CTMC.

Definition 2.6. *The embedded DTMC* $\mathcal{D}$ *of a CTMC* $\mathcal{C} = (S, \boldsymbol{R}, \pi_0, AP, L)$ *is the DTMC* $emb(\mathcal{C}) = (S, \boldsymbol{P}^{emb(\mathcal{C})}, \pi_0, AP, L))$ *where for* $s, s' \in S$

$$P^{emb(\mathcal{C})}(s, s') = \begin{cases} \frac{\boldsymbol{R}(s,s')}{E(s)} & \text{if } E(s) \neq 0 \\ 1 & \text{if } E(s) = 0 \text{ and } s = s' \\ 0 & \text{otherwise} \end{cases} \tag{2.6}$$

Using this definition we can consider the behaviour of the CTMC as staying in a state s for a time exponentially distributed with rate $E(s)$ after which time an instantaneous transition will be made to state s' with probability given by $P^{emb(\mathcal{M})}(s, s')$.

In order to allow for the analysis of the CTMC we also define the infinitesimal generator for the CTMC.

Definition 2.7. *The infinitesimal generator matrix for the CTMC* $\mathcal{C} = (S, \boldsymbol{R}, \pi_0, AP, L)$ *is the matrix* $\mathcal{Q} : S \times S \to \mathbb{R}$ *defined as:*

$$\mathcal{Q}(s, s') = \begin{cases} \boldsymbol{R}(s, s') & \text{if } s \neq s' \\ -\sum_{s'' \neq s} \boldsymbol{R}(s, s'') & \text{otherwise} \end{cases} \tag{2.7}$$

Using the solution of the Kolmogorov forward equations for the Markov chain [7] allows us to calculate the probability distribution at time t as:

$$\boldsymbol{\pi}(t) = \boldsymbol{\pi}_0 e^{\mathcal{Q}t} = \boldsymbol{\pi}_0 \left(I + \sum_{n=1}^{\infty} \frac{\mathcal{Q}^n t^n}{n!} \right) \tag{2.8}$$

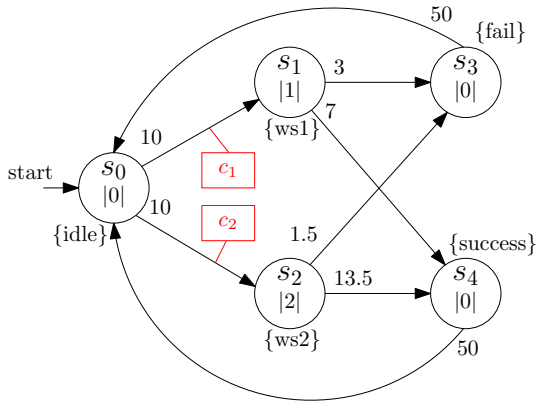


Figure 2.3: CTMC annotated with rewards and costs

2.2.2.2 Extending CTMCs with rewards

Costs and rewards may be associated with a CTMC in a similar way to that described for DTMCs. Unlike a DTMC however the state rewards are calculated with reference to the amount of time a model stays within that state.

Definition 2.8. *Given a CTMC $\mathcal{C} = (S, \boldsymbol{R}, \pi_0, AP, L)$, a reward structure of $\mathcal{C}$ is a pair of real-valued functions $(\underline{\rho}, \iota)$ where:*

- $\underline{\rho} : S \rightarrow \mathbb{R}_{\geq 0}$ *: is a state reward function that defines the rate at which the reward is accumulated whilst the system remains in each state.*
- $\iota : S \times S \rightarrow \mathbb{R}_{\geq 0}$ *: is a transition reward function that defines the value associated with each transition in the model.*

Example 2.3. *Consider the system described in Example 2.1 this time modelled in the continuous domain as shown in Figure 2.3. We assume a time unit of seconds and label each edge with the rate of transition between states. From our idle state we would expect to see the system to move to the next state with a rate of 20 (Eqn. 2.5) i.e. the mean time spent in state s_0 is $\frac{1}{20}s$.*

Two reward structures are applied to the model. A reward is associated with each state such representing the power consumed whilst in the state and this is calculated as the product of the time spent in the state and the reward value associated with the state. In this way state s_2 is seen to consume twice as much power as s_1 per second. The second reward represents the monetary cost of calling a service and is attached to the transitions with value c_1 for s_1 and c_2 for s_2.

The CTMC is then:

$$\begin{aligned} S &= \{s_0, s_1, s_2, s_3, s_4\} \\ \boldsymbol{\pi}_0 &= [1, 0, 0, 0, 0] \\ AP &= \{idle, ws1, ws2, fail, success\} \end{aligned}$$

$$\begin{aligned} L(s_0) &= \{idle\} \\ L(s_1) &= \{ws1\} \\ L(s_2) &= \{ws2\} \\ L(s_3) &= \{fail\} \\ L(s_4) &= \{success\} \end{aligned}$$

where the system matrices are

$$\boldsymbol{R} = \begin{pmatrix} 0 & 10 & 10 & 0 & 0 \\ 0 & 0 & 0 & 3 & 7 \\ 0 & 0 & 0 & 1.5 & 13.5 \\ 50 & 0 & 0 & 0 & 0 \\ 50 & 0 & 0 & 0 & 0 \end{pmatrix} \qquad P^{emb(\mathcal{C})} = \begin{pmatrix} 0 & 0.5 & 0.5 & 0 & 0 \\ 0 & 0 & 0 & 0.3 & 0.7 \\ 0 & 0 & 0 & 0.1 & 0.9 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$\mathcal{Q} = \begin{pmatrix} -20 & 10 & 10 & 0 & 0 \\ 0 & -10 & 0 & 3 & 7 \\ 0 & 0 & -15 & 1.5 & 13.5 \\ 50 & 0 & 0 & -50 & 0 \\ 50 & 0 & 0 & 0 & -50 \end{pmatrix}$$

and the reward structures are defined as

$$\iota = \begin{pmatrix} 0 & c_1 & c_2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \qquad \underline{\rho}^{power} = (0, 1, 2, 0, 0)$$

2.2.2.3 Analysing properties of a CTMC

The properties of a CTMC $\mathcal{C}$ are analysed over its set of finite and infinite paths $Paths^{\mathcal{C}}$. A *finite path* is a sequence $s_1 t_1 s_2 t_2 \ldots s_{k-1} t_{k-1} s_k$, where $s_1, s_2, \ldots, s_k \in S$, $\boldsymbol{\pi}(s_1) > 0$, s_k is an absorbing state, and, for all $i = 1, 2, \ldots, k-1$, $\mathbf{R}(s_i, s_{i+1}) > 0$ and $t_i > 0$ is the time spent in state s_i. An *infinite path* from $Paths^{\mathcal{C}}$ is an infinite sequence $s_1 t_1 s_2 t_2 \ldots$ where $\boldsymbol{\pi}(s_1) > 0$, and, for all $i \geq 1$, $s_i \in S$, $\mathrm{R}(s_i, s_{i+1}) > 0$ and the time spent in state s_i is $t_i > 0$. For any path $\omega \in Paths^{\mathcal{C}}$, the state occupied by the path at time $t \geq 0$ is denoted $\omega@t$. For infinite paths, $\omega@t = s_i$, where i is the smallest index for which $t \leq \sum_{j=1}^{i} t_j$. For finite paths, $\omega@t$ is defined similarly if $t \leq \sum_{j=1}^{k-1} t_j$, and $\omega@t = s_k$ otherwise. Finally, the i-th state on the path ω is denoted $\omega[i]$, where $i \in \mathbb{N}_{>0}$ for infinite paths and $i \in \{1, 2, \ldots, k\}$ for finite paths.

Just as PCTL allows for the definition of properties to evaluate DTMC models, Continuous Stochastic Logic (CSL) is a branching time logic used to specify properties of a CTMC models.

Building on the work of Hannson and Jonsson [28] Aziz first defined Continuous Stochastic Logic (CSL) [36] and this work was further extended by Baier et al. to include a time bounded until operator as well as a steady state operator [37].

Definition 2.9. *The syntax of CSL is defined as follows:*

$$\begin{aligned} \Phi &::= true \mid a \mid \Phi \wedge \Phi \mid \neg\Phi \mid P_{\bowtie p}[\Psi] \mid S_{\bowtie p}[\Phi] \\ \Psi &::= X\Phi \mid \Phi U^{I} \Phi \end{aligned}$$

and the cost/reward augmented CSL state formulae are defined as:

$$R_{\bowtie r}[C^{\leq T}] \mid R_{\bowtie r}[I^{=T}] \mid R_{\bowtie r}[F\ \Phi] \mid R_{\bowtie r}[S]$$

where

- $a \in AP$ *is an atomic proposition*
- $\bowtie\ \in \{<, \leq, \geq, >\}$ *is a relational operator*
- $p \in [0,1]$ *is a probability bound or threshold*
- $r \in \mathcal{R}_{\geq 0}$ *is a reward bound*
- $I \subseteq \mathbb{R}_{\geq 0}$ *and* $T \in \mathbb{R}_{\geq 0}$ *are time interval and time instant respectively*
- Φ *and* ϕ *denote state and path formulae respectively.*

The semantics of CSL are defined with a satisfaction relation $\models$ over the states $s \in S$ and the paths $\omega \in Paths^{\mathcal{C}}$ of a CTMC [38]. CSL semantics are defined recursively by:

$$\begin{array}{lll}
s \models true & \text{for all } s \in S & \\
s \models a & \text{iff} & a \in L(s) \\
s \models \Phi_1 \wedge \Phi_2 & \text{iff} & s \models \Phi_1 \wedge s \models \Phi_2 \\
s \models \neg\Phi & \text{iff} & \neg(s \models \Phi) \\
s \models P_{\bowtie p}[\Psi] & \text{iff} & Pr_s\{\omega \in Paths^{\mathcal{M}} \mid \omega \models \Psi\} \bowtie p \\
\omega \models X\Phi & \text{iff} & \omega = s_1 t_1 s_2 \ldots \wedge s_2 \models \Phi \\
\omega \models \Phi_1 U^I \Phi_2 & \text{iff} & \exists t \in I.(\forall t' \in [0,t).\, \omega@t' \models \Phi_1) \wedge \omega@t \models \Phi_2
\end{array}$$

where a formal definition for the probability measure Pr_s on paths starting in state s is available in [27, 38]. Finally, a state s satisfies a steady-state formula $S_{\bowtie p}[\Phi]$ iff, having started in state s, the probability of the CTMC being in a state where Φ holds *in the long run* satisfies the bound '$\bowtie p$'.

The shorthand notation $\Phi_1 U \Phi_2 \equiv \Phi_1 U^{[0,\infty)} \Phi_2$ and $F^I \Phi \equiv \mathsf{true} U^I \Phi$ is used when $I = [0,\infty)$ in an until formula and when the first part of an until formula is true, respectively. Probabilistic model checkers also support CSL formulae in which the bound '$\bowtie p$' from $P_{\bowtie p}[\Psi]$ is replaced with '=?', to indicate that the computation of the actual bound is required. We distinguish between the probability $Pr_s\{\omega \in Paths^{\mathcal{M}} \mid \omega \models \Psi\}$ that Ψ is satisfied by the paths starting in a state s, and the probability

$$\begin{array}{c}
P_{=?}[\Psi] = \sum_{s \in S} \boldsymbol{\pi}_0(s) Pr_s\{\omega \in Paths^{\mathcal{M}} \mid \omega \models \Psi\} \\
= Pr_{\boldsymbol{\pi}_0}\{\omega \in Paths^{\mathcal{M}} \mid \omega \models \Psi\}
\end{array}$$

that Ψ is satisfied by the CTMC.

The formal semantics associated with CSL when the CTMC is extended with rewards are then:

$$\begin{array}{lll}
s \vDash R_{\bowtie r}[C^{\leq t}] & \text{iff} & Exp^{\mathcal{C}}(s, Y_{C \leq t}) \bowtie r \\
s \vDash R_{\bowtie r}[I^{=t}] & \text{iff} & Exp^{\mathcal{C}}(s, Y_{I=t}) \bowtie r \\
s \vDash R_{\bowtie r}[F\Phi] & \text{iff} & Exp^{\mathcal{C}}(s, Y_{F\Phi}) \bowtie r \\
s \vDash R_{\bowtie r}[S] & \text{iff} & lim_{t \to \infty}(\frac{1}{t}\ Exp^{\mathcal{C}}(s, Y_{C \leq t})) \bowtie r
\end{array}$$

where $Exp^{\mathcal{C}}(s, Y)$ denotes the expectation of the random variable Y with respect to the probability measure Pr_s and for any path ω [27].

Table 2.2: Examples of CSL formulae

Description	CSL formula
The probability of reaching the *fail* state within the first T seconds is less than 0.2	$P_{<0.2}\ [F^{[0,T]}\ fail\ ]$
The probability of reaching the *success* state within T seconds without first visiting the *fail* state is at least 0.8	$P_{\geq 0.80}\ [\ \neg\ fail\ U^{[0,T]}\ success\ ]$
The expected cost to reach the *success* state within T seconds will be at most 50	$R^{cost}_{\leq 50}\ [\ F^{[0,T]}\ success\ ]$

Example 2.4. *If we consider the CTMC presented in Example 2.3 then we can analyse the model against a set of system requirements by encoding them in CSL as shown in Table 2.2.*

2.3 Phase-type distributions

Whilst individual states within a CTMC are modelled with exponential holding times more general stochastic distributions may be obtained through the composition of states using phase-type distributions (PHDs) [39]. PHDs model stochastic processes where the event of interest is the time to reach a specific state and are widely used in performance modelling of systems from domains ranging from call centres to healthcare[40],[41],[42].

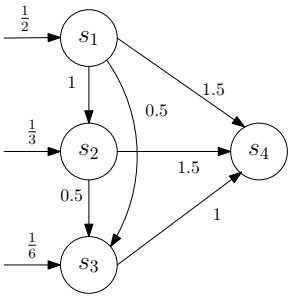
PHDs support efficient numerical and analytical evaluation [39], and can approximate arbitrarily closely any continuous distribution with a strictly positive density in $(0, \infty)$ [43], although PHD fitting of distributions with deterministic delays requires extremely large numbers of states.

Definition 2.10. *A PHD is defined as the distribution of the time to absorption in a CTMC with one absorbing state [39]. The $N \geq 1$ transient states of the CTMC are called the* phases *of the PHD. With the possible reordering of states, the infinitesimal generator matrix of this CTMC can be expressed as:*

$$\mathbf{Q} = \left[\begin{array}{c|c} \mathbf{D_0} & \mathbf{d_1} \\ \hline \mathbf{0} & 0 \end{array}\right], \tag{2.9}$$

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 of zeros, 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, so we additionally have $\mathbf{D_0 1} + \mathbf{d_1} = \mathbf{0}$, where $\mathbf{1}$ and $\mathbf{0}$ are column vectors of N ones and N zeros, respectively. Thus, $\mathbf{d_1} = -\mathbf{D_0 1}$ 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 $\boldsymbol{\pi}_0$ (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.

Example 2.5. *Consider the CTMC described in Figure 2.4 which has a single absorbing state (s_4).*



$$\mathbf{Q} = \begin{pmatrix} -3 & 1 & 0.5 & 1.5 \\ 0 & -2 & 0.5 & 1.5 \\ 0 & 0 & -1 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Figure 2.4: A CTMC with a single absorbing state and the associated infinitesimal generator matrix

Since $\mathbf{d_1}$ *is implicitly given by* $\mathbf{D}_0$ *Then the PHD is the fully defined by:*

$$\mathbf{D}_0 = \begin{bmatrix} -3 & 1 & 0.5 \\ 0 & -2 & 0.5 \\ 0 & 0 & -1 \end{bmatrix} \tag{2.10}$$

$$\pi_0 = \begin{bmatrix} \frac{1}{2} & \frac{1}{3} & \frac{1}{6} \end{bmatrix} \tag{2.11}$$

2.3.1 Erlang distributions

The Erlang distribution [7] was developed by A.K. Erlang to investigate telephone system performance. An Erlang distribution is a form of PHD in which k exponential phases with the same rate parameter (λ) are placed in series as shown in Figure 2.5.

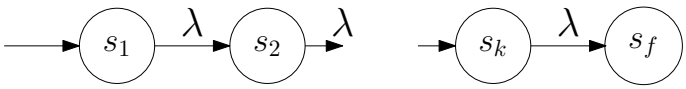


Figure 2.5: CTMC of an Erlang distribution with n phases

The Erlang distribution is then denoted $E(k, \lambda)$ where the initial probability vector is $\pi_0 = \begin{bmatrix} 1, & 0, & \cdots, & 0 \end{bmatrix}$ such that the system always starts in state s_1 and traverses all successive states until it reaches the absorbing state s_f. The distribution then represents the expected time to reach the absorbing state and has a probability density function (PDF) given by

$$f(x) = \frac{\lambda^k}{(k-1)!} x^{k-1} e^{-\lambda x} \text{ for } x \geq 0 \tag{2.12}$$

such that $f(x) = \lambda e^{-\lambda x}$ for $x \geq 0$ represents the density of the exponential function when $k = 1$.

The cumulative distribution function (CDF) of an Erlang function is then defined by

$$F(x) = 1 - \sum_{i=0}^{k-1} \frac{(\lambda x)^i}{i!} e^{-\lambda x} \text{ for } x \geq 0 \tag{2.13}$$

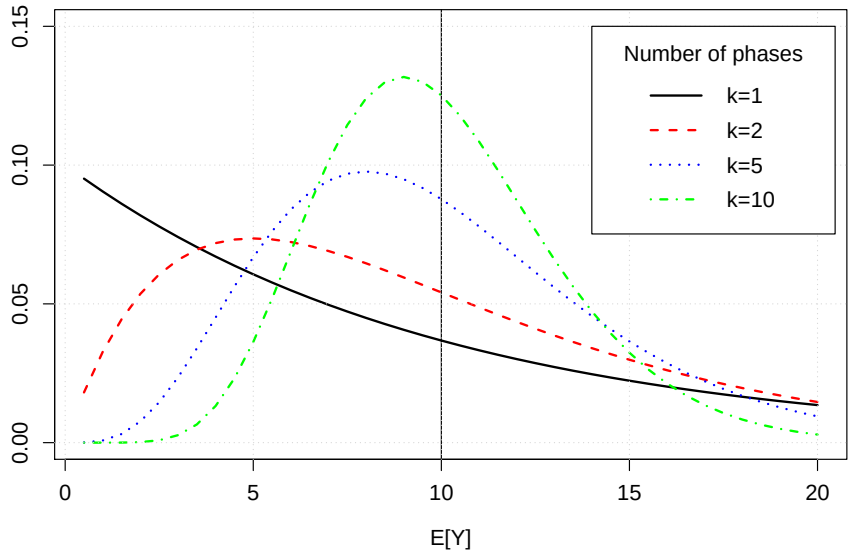


Figure 2.6: Probability density function for an Erlang distribution as the number of phases increases

A random variable Y with an Erlang distribution then has an expected (or mean) time to reach the absorbing state of:

$$E[Y] = \frac{k}{\lambda} \tag{2.14}$$

and the variance is

$$\text{VAR}[Y] = \frac{k}{\lambda^2} \tag{2.15}$$

Figure 2.6 shows the probability density function (PDF) for a set of Erlang distributions where $k \in \{1, 2, 5, 10\}$ and $E[Y] = 10$. It can be seen that as k increases the peak in the PDF becomes more pronounced with the center of the peak moving towards the expected holding time.

We note that an Erlang distribution with mean m has variance $\frac{m^2}{k}$ and thus tends to be deterministic as $k \to \infty$. Whilst PHDs can approximate any continuous distribution, the modelling of distributions with regions of zero density is difficult and [44]. Indeed it has been shown that Erlang distributions are the be PHD for a fixed-delay distribution [45].

2.3.2 Hyper-Erlang distributions

Hyper-Erlang distributions [46],[47] are a class of phase-type distribution in which a mixture of n mutually independent Erlang distributions are weighted with an initial probability distribution $\pi_0 = [\alpha_1, \alpha_2, \cdots \alpha_n]$. This is shown graphically in Figure 2.7 where each branch is an Erlang distribution.

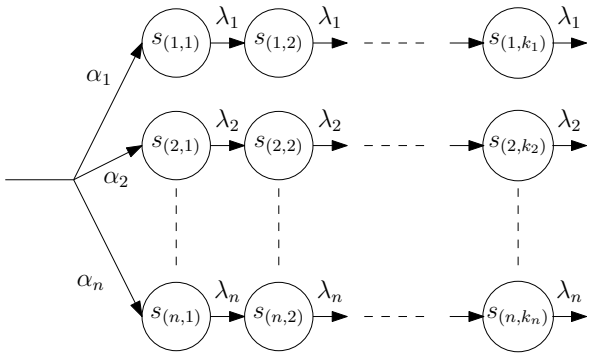


Figure 2.7: Hyper-Erlang distribution

If k_i is the number of phases in the i^{th} Erlang distribution then the PDF is [39]

$$f(x) = \sum_{i=1}^{n} \pi_i \frac{(\lambda_i x)^{k_i - 1}}{(k_i - 1)!} \lambda_i e^{-\lambda_i x} \text{ for } x \geq 0 \tag{2.16}$$

where π_i is the i^{th} element of the initial distribution vector π_0. The CDF is then given by

$$F(x) = 1 - \sum_{i=1}^{n} \pi_i \sum_{j=0}^{k_i - 1} \frac{(\lambda_i x)^j}{j!} e^{-\lambda_i x} \text{ for } x \geq 0 \tag{2.17}$$

The state space of the Hyper-Erlang has $\sum_{i=1}^{n} k_i$ transient states and one absorbing state. When $n = 1$ then the Hyper-Erlang is a simple Erlang distribution. The continuous time Markov chain of a Hyper-Erlang can be described by an infinitesimal generator matrix $\mathbf{Q}$ and the initial distribution vector where

$$\mathbf{Q} = \begin{bmatrix} \mathbf{Q}_1 & 0 & \cdots & 0 \\ 0 & \mathbf{Q}_2 & \cdots & 0 \\ 0 & 0 & \cdots & \mathbf{Q}_n \end{bmatrix} \tag{2.18}$$

and $\mathbf{Q}_i$ represents the infinitesimal generator matrix of the i^{th} Erlang branch.

2.4 Probabilistic model checking software

The mathematical foundations of model checking have long been understood but the practical application of these techniques only became possible with the creation of suitable model checking tools. The first such tool was developed by Clarke et al. [48],[49] and utilised temporal logics, in computational tree logic, to specify system properties against which a finite state system model could be verified. The same team went on to develop a symbolic model checker [50], SMV, which was able of handling large real-world problems and provide

counter examples where properties were violated.

The primary focus of SMV was the verification of hardware, e.g. integrated circuits, however, SPIN [51], developed by Bell labs, was targeted at the "efficient verification of multi-threaded software, not the verification of hardware circuits". Again this tool was able to demonstrate practical applicability and was successfully used to model control systems for flood defences as well as mission critical systems for space exploration.

Whilst these tools provided support for a wide range of problem domains they were restricted to finite state systems in which the transitions where deterministic. For a wide class of problems however the transition between states is unreliable or unpredictable and therefore the analysis of such problems requires the use of probabilistic model checking.

A model checker such as SMV or SPIN takes a state-transition model and a property described in temporal logics and returns either "true" or "false". A counterexample is also returned should the property fail to be met. By contrast a probabilistic model checker accepts a probabilistic model (typically a Markov chain) that encodes the chance of a transition occurring as a probability. Probabilistic temporal logics are then employed to define system specifications and the results are returned as a likelihood.

Even though the algorithms for probabilistic model checking were first proposed in the 1980s [52], the first "industrial strength model checking tools" were not available until the 2000s [52]. Today there are numerous such tools which have been applied in a wide range of contexts. Jansen et al, in their 2008 paper [53] compared five tools to assess their strengths and weaknesses. PRISM [54],[8] performs well in this study and is a widely used tool which provides an easy to use user interface and a modelling language which is now well supported in newer tools such as Storm [11]. For this reason PRISM was chosen as the tool of choice for probabilistic model checking within this work.

Model checking, and probabilistic model checking, have been successfully applied in the analysis of real-world problems; however, challenges still remain [55]. In particular quantifying many real-world behaviours is difficult and existing tools cannot support the design and verification of real systems unless the analysed models are accurate representations of those behaviours. Ensuring the accuracy of performance models, therefore, remains a major challenge.

Part II

Continuous Time

Chapter 3

Observation-Enhanced CTMC Refinement

In this chapter we present a new method for the accurate analysis of transient quality of service (QoS) properties of operational processes. Our method takes as input a high-level abstract continuous-time Markov chain (CTMC) model of the process to be analysed. Each state in the high-level model represents an activity or task in the operational process. Each state in the model is then refined, i.e. the accuracy of the state is improved, through the addition of additional states which more accurately represent the execution times observed for each task. The refined CTMC can then be analysed with existing probabilistic model checkers to accurately predict the value of QoS properties.

Modern systems and processes are often constructed using complex interconnected units [56]. The performance, cost, resource use and other QoS properties of these underpin important engineering and business decisions. As such QoS analysis has been the subject of intense research [57],[58],[59],[60]. The solutions devised by this research can analyse a broad range of QoS properties by using *performance models* such as Petri Nets [61],[62], layered queuing networks [63], Markov chains [64],[30] and timed automata [65], together with tools for their simulation (e.g. Palladio [66] and GreatSPN [67]) and formal verification (e.g. PRISM [8] and UPPAAL [68]).

These advances enable the effective analysis of many types of performance models. However, they cannot support the design and verification of real systems unless the analysed models are accurate representations of the real-world behaviour, and ensuring the accuracy of performance models remains a major challenge. We address this challenge for *continuous-time Markov chains* (CTMCs) and introduce a tool-supported method for Observation-based Markov chaiN refInement (OMNI) and accurate QoS analysis of operational processes.

The OMNI method comprises the five activities shown in Figure 3.1. The key characteristic of OMNI is its use of observed execution times for the operational activities of the analysed process to refine a high-level abstract CTMC whose states correspond to individual activities. As such, the first OMNI step is the collection of these execution time observations, which can come from unit testing activities prior to system integration, from logs of other systems that use the same activities, or from logs of the analysed process. The second OMNI step

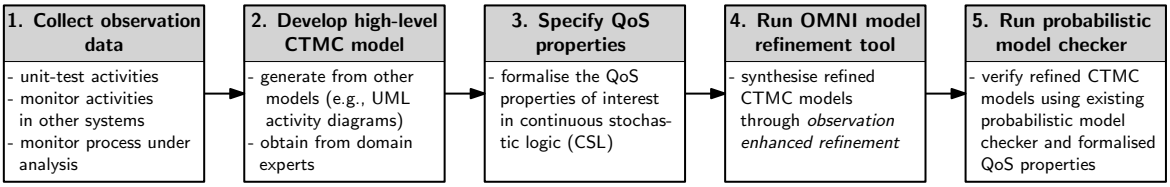


Figure 3.1: OMNI workflow for the QoS analysis of operational processes

involves the development of a high-level CTMC model of the process under analysis. This model can be generated from more general models such as annotated UML activity diagrams as in [31],[69], or can be provided by the domain experts. The next OMNI step requires the formalisation of the QoS properties of interest as continuous stochastic logic formulae.

In the fourth step of our OMNI method we use observed activity execution times to refine the high-level abstract CTMC where states correspond with operational activities. The refinement of model states in OMNI consists of two parts. The first part makes the CTMC more realistic through the addition of states and transitions that model the non-zero minimum execution times associated with operational activities. We use additional states and transitions corresponding to Erlang Distributions for this purpose. The CTMC is then further refined by using phase-type distributions to model the variation in execution times observed in execution logs. The resulting OMNI-refined CTMCs model the execution times of operational activities with much greater accuracy than traditional CTMC modelling techniques.

In the last activity of our method, the refined CTMC models generated by OMNI are analysed with established probabilistic model checkers such as PRISM [8] and STORM [11]. These models support the accurate and efficient analysis of a broad spectrum of QoS properties specified in continuous stochastic logic [36] and, as such, OMNI's observation-enhanced QoS analysis can prevent many of the invalid engineering and business decisions associated with traditional CTMC-based QoS analysis.

Several factors can impede or impact the success of our OMNI method:

1. Operational activities with execution times that are not statistically independent. Markov models assume that the transition rates associated with different states are statistically independent. If the execution times of different activities are not independent, then this premise is not satisfied, and OMNI cannot be applied. For example a process composed of two tasks requires that the time taken to complete task one has no effect on the time taken to complete task two. For some applications this is not the case. The time taken to write a program and the time taken to test it may, for example, be correlated.
2. Changing behaviour. If the temporal characteristics of an activity change significantly over time, then OMNI cannot predict the changed behaviour. This is a more general difficulty with model-based prediction. When the real-world system changes in a predictable manner this may be accounted for through the modification of model structures or parameters. For example the time taken for a web-service to respond may be a function of the load on the server. If we know that the load is high at certain times of day then we may construct a model which uses different characteristics during this period. Where the nature of the change is unknown, however, the model, and therefore its analysis, will be inaccurate.

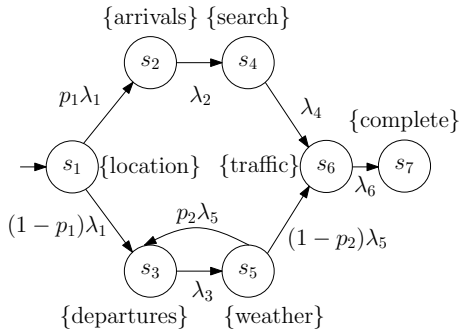


Figure 3.2: High-level abstract CTMC modelling of a web based travel application

3. Insufficient observations of execution times. The accuracy of OMNI-refined models decreases when fewer observations of the system components are available. A task which is characterised by a complex non-parametric distribution will require many data points to describe it. Indeed for small sample sets we may not observe certain modes of operation at all. We provide details about the impact of the training dataset size on the OMNI accuracy in Section 5.2.4.

The remainder of this chapter is structured as follows. Section 3.1 describes the web process which we use to evaluate OMNI, as well as to motivate and illustrate our QoS analysis method. The method employed for state refinement is then presented in Section 3.2. Finally Section 3.3 provides an overview of related work.

3.1 Motivating example: QoS analysis of a web application

To illustrate the limitations of traditional CTMC-based QoS analysis, we consider a web application that enacts two processes in response to the following requests:

1. Requests from users who plan to meet and entertain a visitor arriving by train.
2. Requests from users looking for a possible destination for a day trip by train.

The handling of these requests by the application is modelled by the high-level abstract CTMC from Figure 3.2, which can be obtained from a UML activity diagram of the application. The initial state s_1 of the CTMC corresponds to finding the location of the train station. For the first request type, which is expected to occur with probability p_1, this is followed by finding the train arrival time (state s_2), identifying suitable restaurants in the area (state s_4), obtaining a traffic report for the route from the user's location to the station (state s_6), and returning the response to the user (state s_7).

For the second request type (day trips), which occurs with probability $1 - p_1$, state s_1 is followed by finding a possible destination (state s_3), and obtaining a weather forecast for this destination (state s_5). With a probability of p_2 the weather is unsuitable and a new destination is selected (back to state s_3). Once a suitable destination is selected, the traffic report is obtained for travel to the station (state s_6) and the response is returned to the user

Table 3.1: Web services considered for the web 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`	1.85
weather	WebserviceX.net weather service	`http://www.webservicex.net/globalweather.asmx`	1.11
traffic	Bing traffic service	`http://dev.virtualearth.net/REST/v1/Traffic`	2.51

(state s_7).

Each activity in the processes is implemented using a web service component. The execution rates λ_1 to λ_6 depend on the implementations used for these components. We consider that a team of software engineers wants to decide if the real web services from Table 3.1 are suitable for building the application. If they are suitable, the engineers need:

1. To select appropriate request-handling times to be specified in the application service-level agreement (SLA);
2. To choose a pricing scheme for the application.

Accordingly, the engineers want to assess several QoS properties of the travel application variant built using these publicly available web 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 QoS analysis based on CTMC (as well as queueing network) models. In order to estimate the rate parameter of the exponential distribution we make use of the maximum likelihood estimator. Derivation of the MLE for an exponential distribution is provided in Appendix A. For the observed service execution times $t_{i1}, \ldots, t_{in}$ for service i the estimate for the service rate λ_i is

$$\lambda_i = \left(\frac{t_{i1} + t_{i2} + \cdots + t_{in}}{n} \right)^{-1}. \tag{3.1}$$

These execution times can be taken from existing logs (e.g. of other applications that use the same services) or can be obtained through testing the web services individually. Finally, a probabilistic model checker is used to analyse properties **P1**–**P3** of the resulting CTMC.

For this purpose, the three properties are first formalised as transient-state CSL formulae:

$$\begin{array}{ll} \mathbf{P1} & P_{=?}[F^{[0,T]}\,complete] \\ \mathbf{P2} & P_{=?}[\neg arrivals\ U^{[0,T]}\,complete]/(1-p_1) \\ \mathbf{P3} & P_{=?}[F^{[0,T]}\,complete] - 2 \cdot P_{=?}[F^{(3,\infty)}\,complete] \end{array} \tag{3.2}$$

The value of T to be specified in the SLA is unknown a priori and hence we evaluate each property for a range of T values where $0 < T \leq 4$ for **P1** and **P2**, and $0 < T \leq 3$ for **P3**.

To replicate this process, we implemented a prototype version of the application and we used it to handle 270 randomly generated requests for $p_1 = 0.3$ and $p_2 = 0.1$. Obtaining transition probabilities for Markov chains from real-world systems, and the effects of transition probabilities on system performance, have previously been considered [70],[71]. To decouple these effects from those due to the temporal characteristics of component behaviours, we utilise fixed probabilities for our motivating example. We obtained sample execution times for each web service (between 81 for arrivals and search and 270 for location and traffic), and we applied (3.1) to these observations, calculating the estimated service rates from Table 3.1. We then used the model checker PRISM to analyse the CTMC for these rates, and thus to predict the values of properties (3.2).

To assess the accuracy of the predictions, we also calculated the actual values of these properties at each time value T using detailed timing information logged by our application. The error associated with a single property evaluation may be quantified as the absolute difference between actual and predicted values

$$|actual(T) - predicted(T)| \tag{3.3}$$

The predictions obtained through CTMC analysis and the actual property values across the range of T values are compared in Figure 3.3. The errors reported in the figure are calculated using the distance measure recommended for assessing the overall error of CTMC/PHD model fitting in [72],[39],[73],[74], i.e., the area difference between the actual and the predicted property values:

$$error = \int_0^{T_{\max}} |actual(T) - predicted(T)|\ dT, \tag{3.4}$$

where $T_{\max} = 4$ for properties **P1** and **P2**, and $T_{\max} = 3$ for property **P3**.[1]

Later in the chapter, we will use this error measure to assess the improvements in accuracy due to the OMNI model refinement. In this section we focus on the limitations of CTMC-based transient analysis. Therefore, recall that the software engineers must make their decisions based only on the predicted property values from Figure 3.3; two of these decisions and their associated scenarios are described below.

Scenario 1. The engineers note that:

[1] Both underestimation and overestimation of QoS property values contribute to the error because both can lead to undesirable false positives or false negatives when assessing whether QoS requirements are met. For example, overestimates of the overall success probability of a system can falsely indicate that a requirement that places a lower bound on this probability is met and the system is safe to use (false negative), while underestimates of the same property can falsely indicate that the requirement is violated and the system should not be used (false positive).

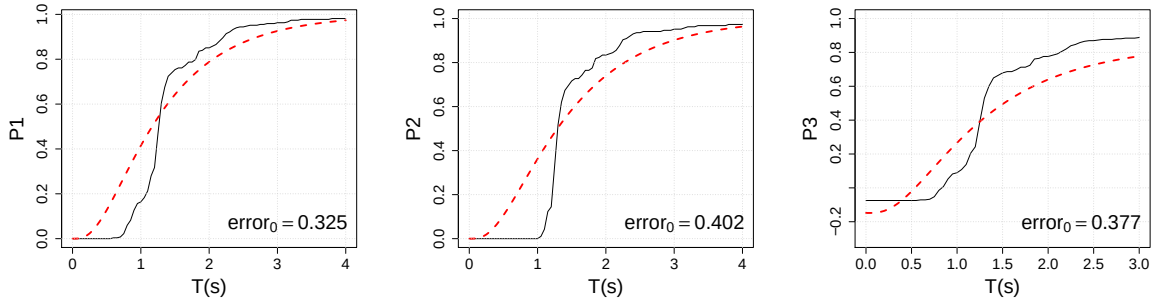


Figure 3.3: Predicted (dashed lines) versus actual (continuous lines) property values

(a) the predicted overall success probability (property **P1**) at T=1s is 0.415 (marked 1a in Figure 3.4), i.e., slightly over 40% of the requests are predicted to be handled within 1s;

(b) the predicted day-trip success probability (property **P2**) at $T = 1$s is 0.363 (1b in Figure 3.4), i.e., over 36% of the day-trip requests are predicted to be handled within 1s;

(c) the expected profit (property **P3**) at $T = 1$s, i.e., when charging 1 cent for requests handled within 1s, is 0.27 cents (1c in Figure 3.4).

Accordingly, the engineers decide to use the services from Table 3.1 to implement the travel web application, with an SLA "promising" that requests will be handled within 1s with 0.4 success probability, "day trip" requests will be handled within 1s with 0.35 success probability, and charging 1 cent for requests handled within 1s. As shown in Figure 3.4, the actual property values at $T = 1$s are 0.164 for **P1** (marked 1a$'$ in Figure 3.4), 0 for **P2** (1b$'$ in Figure 3.4) and 0.09 cents for **P3** (1c$'$ in Figure 3.4), so this decision would be wrong – both promises would be violated by a wide margin, and the actual profit would be under a third of the predicted profit.

Scenario 2. The engineers observe that the success probabilities of handling requests or "day trip" requests within 2s are below 0.8 – the predicted values for properties **P1** and **P2** at $T = 2$s are 0.79 (2a in Figure 3.3) and 0.74 (2b in Figure 3.3), respectively; and/or that the expected profit is below 0.7 cents per request when charging 1 cent for each request handled within 2s (2c in Figure 3.3). As such, they decide to look for alternative services for the application. As shown by points 2a'–2c' in Figure 3.3, all the constraints underpinning this decision are actually satisfied, so the decision would also be wrong.

We chose the times and constrains in the two hypothetical decisions to show how the current use of idealised CTMC models in QoS analysis *may* yield invalid decisions. The fact that choosing different times and constrains could produce valid decisions is not enough: engineering decisions are meant to be consistently valid, not down to chance. It is this major limitation of traditional CTMC-based QoS analysis that our CTMC refinement method addresses as described in the next section.

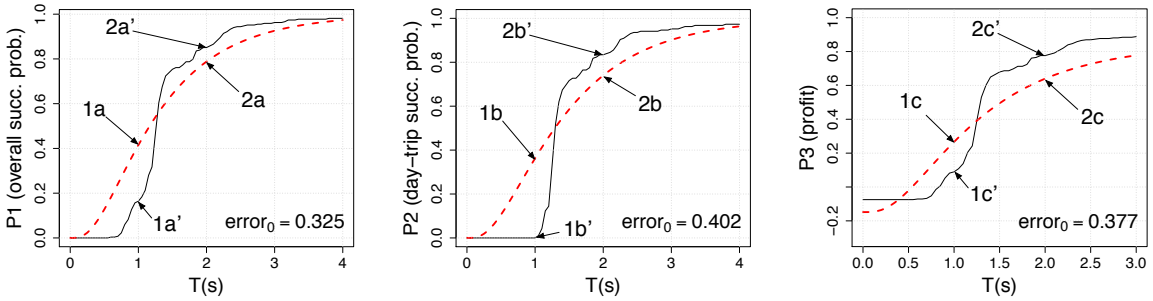


Figure 3.4: SLA property evaluation using CTMC verification

3.2 The OMNI method for CTMC refinement

3.2.1 Overview

OMNI addresses the refinement of high-level CTMC models $\mathsf{CTMC}(S, \boldsymbol{\pi}, \mathbf{R})$ that satisfy the following assumptions:

- Each state $s_i \in S$ corresponds to an activity of the process, and $\boldsymbol{\pi}(s_i)$ is the probability that s_i is the initial activity executed;
- For any distinct states from $s_i, s_j \in S$, the transition rate $\mathbf{R}(s_i, s_j) = p_{ij}\lambda_i$, where p_{ij} represents the (known or estimated) probability (2.6) that activity i is followed by activity j and λ_i is obtained by applying (3.1) to n_i observed execution times $\tau_{i1}, \tau_{i2}, \ldots, \tau_{in_i}$ of activity i;
- Each state $s_i \in S$ is labelled with the name of its corresponding activity, which we will call "activity i" for simplicity.

This CTMC model makes the standard assumption that execution times are exponentially distributed. However, this assumption is typically invalid for two reasons. First, each activity i has a *delay* δ_i (i.e. minimum execution time) approximated by

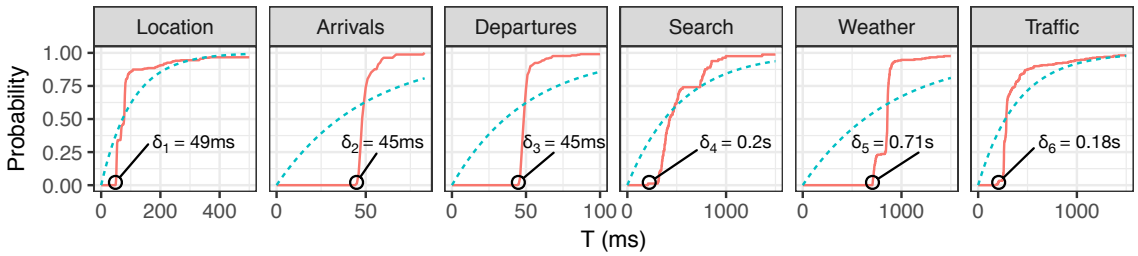
$$\delta_i \approx \min_{j=1}^{n_i} \tau_{ij} \tag{3.5}$$

such that its probability of completion within δ_i time units is zero. In contrast, modelling the execution time of the activity as exponentially distributed with rate λ_i yields a non-zero probability $1 - e^{-\lambda_i \delta_i}$ of completion within δ_i time units. Second, even the *holding times*

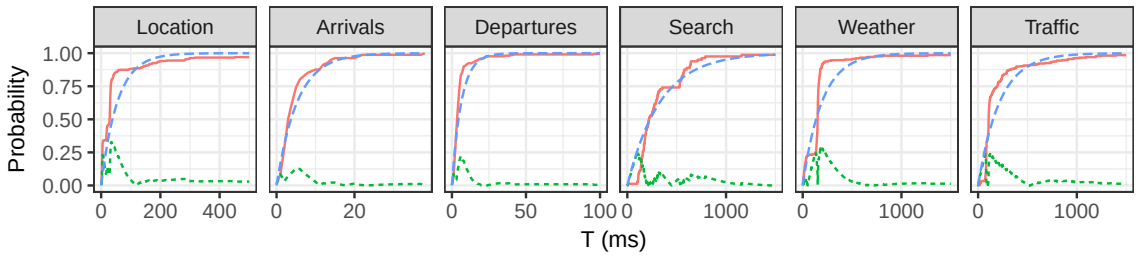
$$\tau'_{i1} = \tau_{i1} - \delta_i,\ \tau'_{i2} = \tau_{i2} - \delta_i,\ \ldots,\ \tau'_{in_i} = \tau_{in_i} - \delta_i \tag{3.6}$$

of the activity are rarely exponentially distributed. An extreme scenario when this is not the case is when an activity is implemented using a web service with instances running on multiple non-identical computers for load balancing purposes: in this scenario, the execution times will have a multimodal distribution.

Example 3.1. *Figure 3.5a shows the empirical cumulative distribution functions (CDFs) for the execution times of the six services from our motivating example (cf. Table 3.1), and the associated exponential models with rates given by (3.1). The six services have minimum*



(a) Empirical CDF for the service execution times (continuous lines) versus exponential models with rates computed from observed data (dashed lines)



(b) Empirical CDF for the service holding times (continuous lines) versus exponential models with rates computed from observed holding times (long dashed lines); for all services except Arrivals the difference between the two (short dashed lines) exceeds 20% for multiple values of T

Figure 3.5: The services from the motivating example have non-zero delays and non-exponentially distributed holding times

observed execution times δ_1 to δ_6 between 45ms and 0.71s (due to network latency and request processing time), and their exponential model is a poor representation of the observed temporal behaviour. Furthermore, the best-fit exponential model of the observed holding times for these services (shown in Figure 3.5b) is also inaccurate.

OMNI overcomes these significant problems, which can lead to erroneous analyses of the transient QoS properties of the modelled system, and thus to invalid verification conclusions and design decisions as shown in our motivating example. To this end, OMNI generates a refined CTMC where each activity is replaced with an Erlang representation of the delay associated with each activity and a phase-type distribution model of the residual holding time for the activity once the delay has been extracted. For our motivating example then the system may be considered as shown in Figure 3.6[2].

3.2.2 Delay modelling

Where an activity exhibits a non-zero execution time OMNI extends the CTMC with additional states and transitions that model the minimum execution times (i.e. *delays*) by means of Erlang distributions, i.e., a sum of several independent exponential distributions with the same rate [7]. Whilst phase-type distribution models can theoretically approximate any continuous distribution arbitrarily closely, regions of zero density (delays) pose a particular problem for PHD fitting algorithms. Erlang models have been shown to be the

[2]The graphical notion introduced in Figure 3.6 is reused in Chapter 4.

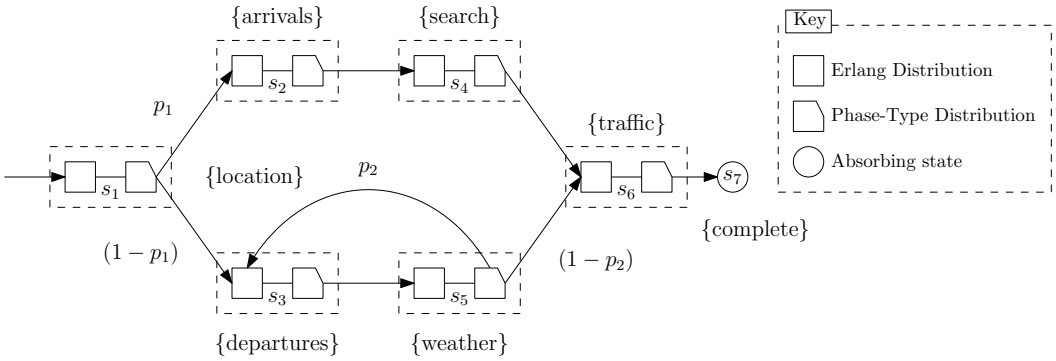


Figure 3.6: Motivating example with delay and holding time abstraction

best possible fit for these delays [44] and OMNI therefore automatically provides for the automatic extraction and modelling of models where they are present in the observed data.

As shown in Figure 3.7, this involves replacing each state s_i with a sequence of delay-modelling states z_{i1}, z_{i2}, ..., z_{ik_i} that encode an Erlang-k_i distribution of rate λ_i^E, followed by a state s_i' with transitions matching those from the high-level CTMC but of rate λ_i'.

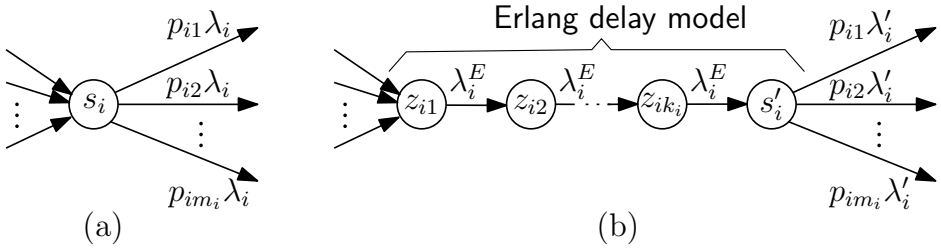


Figure 3.7: Modelling component i in the (a) abstract and (b) refined CTMC

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 z_{ik_i} within $\delta_i(1-\epsilon)$ time units of entering z_{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 modelling. Figure 3.8 shows the cumulative distribution for an Erlang-k distribution for a delay of 10 time units and illustrates the error bound ϵ as k is increased.

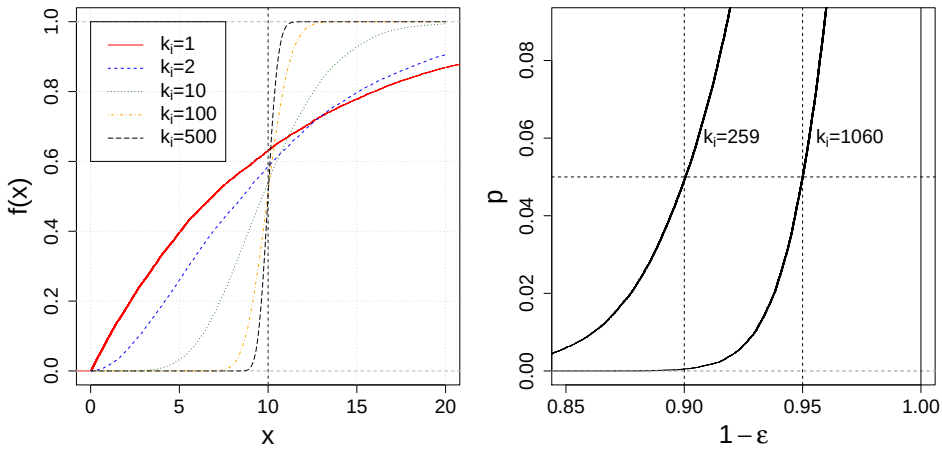
Theorem 1. *Given an error bound $\epsilon \in (0,1)$, if the delay modelling parameters k_i, λ_i^E and λ_i' satisfy*

$$\begin{aligned} &(a)\ 1 - \textstyle\sum_{l=0}^{k_i-1} \frac{(k_i(1-\epsilon))^l e^{-k_i(1-\epsilon)}}{l!} = p \\ &(b)\ \lambda_i^E = \tfrac{k_i}{\delta_i} \\ &(c)\ \lambda_i' = \tfrac{\lambda_i}{1-\lambda_i\delta_i} \end{aligned} \tag{3.7}$$

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

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

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



(a) (b)

Figure 3.8: Erlang distribution: (a) cumulative distribution function for an Erlang-k distribution with rate $\lambda = k/10$ (note that $k = 1$ gives the exponential distribution); and (b) relationship between ϵ, p and k_i from Theorem 1

is also the expected time for the high-level CTMC to leave state s_{iN_i} after entering state s_{i1}, so the delay modelling preserves the first moment of the distribution associated with the refined CTMC states.

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 (3.7a) can be rewritten as $F(k_i, \lambda_i^E, \delta_i(1-\epsilon)) = p$ since $k_i = \lambda_i^E \delta_i$ according to (3.7b). Therefore, the probability that the Erlang delay model from Figure 3.7 will transition from entering state z_{i1} to exiting state z_{ik_i} within $\delta_i(1-\epsilon)$ time units is p. For part (ii), the expected time for the CTMC to leave state s_i' after entering z_{i1} is the sum of the mean of the Erlang-k_i distribution with rate λ_i^E and the mean of the exponential distributions with rate λ_i', i.e.

$$k_i \frac{1}{\lambda_i^E} + \frac{1}{\lambda_i'} = \delta_i + \frac{1-\lambda_i\delta_i}{\lambda_i} = \frac{\delta_i\lambda_i}{\lambda_i} + \frac{1-\lambda_i\delta_i}{\lambda_i} = \frac{1}{\lambda_i}$$

□

Theorem 1 supports the calculation of the delay model parameters for a state $s_i \in S$ as follows:

1. Approximate the delay δ_i for the activity associated with s_i using (3.5).
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 (3.7a) for k_i. This can be done 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 Δ_i, by using precomputed k_i values as in Table 3.2;
3. Calculate λ_i^E and λ_i' using (3.7b) and (3.7c), respectively.

Table 3.2: Precomputed probability of leaving an Erlang delay for a set of k_i values and and error bound ϵ. The highlighted cell corresponds to $p < 0.05$ as required in Example 3.2.

$(1-\epsilon)$	$k_i = 10$	$k_i = 50$	$k_i = 100$	$k_i = 259$	$k_i = 500$
0.80	0.283376	0.070335	0.017108	0.000289	8.11×10^{-7}
0.81	0.295858	0.082033	0.022651	0.000567	2.85×10^{-6}
0.82	0.308481	0.095025	0.029572	0.001069	9.33×10^{-6}
0.83	0.321226	0.109348	0.038085	0.001942	0.000028
0.84	0.334080	0.125024	0.048407	0.003408	0.000081
0.85	0.347026	0.142062	0.060744	0.005776	0.000213
0.86	0.360049	0.160454	0.075287	0.009468	0.000529
0.87	0.373132	0.180174	0.092202	0.015025	0.001230
0.88	0.386260	0.201181	0.111616	0.023107	0.002687
0.89	0.399419	0.223415	0.133612	0.034471	0.005526
0.90	0.412592	0.246802	0.158221	0.049937	0.010717
0.91	0.425765	0.271252	0.185416	0.070318	0.019642
0.92	0.438924	0.296660	0.215109	0.096347	0.034083
0.93	0.452054	0.322910	0.247147	0.128582	0.056105
0.94	0.465142	0.349875	0.281319	0.167321	0.087793
0.95	0.478174	0.377421	0.317357	0.212526	0.130876
0.96	0.491138	0.405404	0.354944	0.263777	0.186282
0.97	0.504021	0.433681	0.393723	0.320267	0.253760
0.98	0.516812	0.462104	0.433311	0.380834	0.331673
0.99	0.529498	0.490528	0.473304	0.444037	0.417054

Example 3.2. *Consider the arrivals state from from our running example which, according to our experimental data, has a delay of $\delta = 45ms$ Suppose that we want to model this delay with an error bound $\epsilon = 0.1$ and a probability $p = 0.05$ (highlighted in Table 3.2). This gives $k = 259$ and the other delay modelling parameters are calculated as:*

$\lambda^E = \frac{k}{\delta} = \frac{259}{0.045} = 5756s^{-1}$,

$\lambda' = \frac{\lambda}{1-\lambda\delta} = \frac{19.88}{1-19.88\cdot0.045} = 188.61s^{-1}$

(where the rate λ is taken from Table 3.1).

Example 3.3. *Consider again the web application from our motivating example, and its three quality properties from (3.2). We applied the OMNI delay extraction to each state associated with a service in the CTMC from Figure 3.2. We set $\epsilon = 0.1$ and $p = 0.05$, so state s_i, $1 \le i \le 6$, from the original CTMC was replaced with an Erlang delay model as in Figure 3.7(b), where $k_i = 259$ and the values for λ_i^E and λ_i' are given in Table 3.3. Accordingly, the refined CTMC had 6×259 more states and transitions than the original CTMC. Figure 3.9 compares the actual values of properties **P1**–**P3** with the values predicted through the analysis of the refined CTMC. As anticipated, the error (3.4) 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 $T = 1.4s$ are 0.565 (instead of an actual value of 0.723) and 0.462 (instead of 0.674), respectively. In Section 3.2.3 we show how OMNI addresses this difference.*

Table 3.3: Erlang delay model parameters for the states of the CTMC from Figure 3.2

s_i	**label**	δ_i(ms)	$\lambda_i^E(\mathrm{s}^{-1})$	λ_i' (s^{-1})	s_i	**label**	δ_i (ms)	$\lambda_i^E(\mathrm{s}^{-1})$	$\lambda_i'(\mathrm{s}^{-1})$
s_1	location	49	5286	18.20	s_4	search	209	1239	3.02
s_2	arrivals	45	5756	188.61	s_5	weather	706	367	5.08
s_3	departures	45	5756	156.56	s_6	traffic	179	1447	4.56

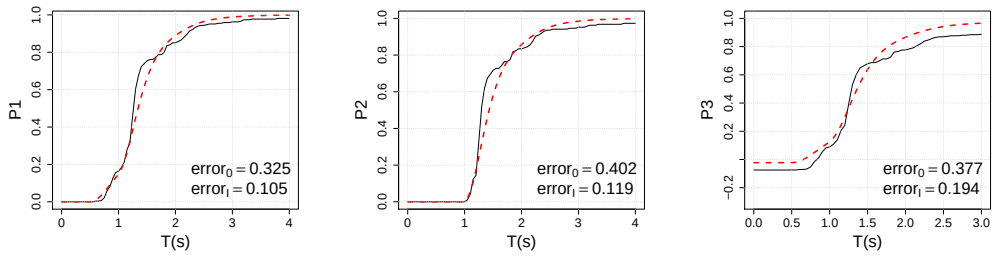


Figure 3.9: Predicted after OMNI delay extraction (dashed lines) versus actual (continuous lines) values for properties **P1**–**P3**, where error_0 and error_I are calculated as in (3.4), before and after delay extraction

3.2.3 Holding-time modelling

Having modelled the delay associated with an activity we now turn our attention to the modelling of residual holding time. For each activity for which we have observation data, we synthesise a phase-type distribution $\mathsf{PHD}(\boldsymbol{\pi}_0, \mathbf{D}_0)$ that models the holding times (3.6), and we replace the relevant state s' of the model $\mathsf{CTMC}(S', \boldsymbol{\pi}', \mathbf{R}')$ obtained after the OMNI delay modelling with this PHD.

Our holding time modelling exploits recent advances in the fitting of phase-type distributions to empirical data. Given the usefulness of PHDs in performance engineering, this area has received considerable attention [39],[75], with effective PHD fitting algorithms developed based on techniques such as moment matching [76],[77], expectation maximisation [78],[79],[80] and Bayes estimation [81],[82]. Recently, these algorithms have been used within PHD fitting approaches that: (a) partition the dataset into segments [80] or clusters [73] 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 the 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 the direct algorithms applied to the entire dataset.

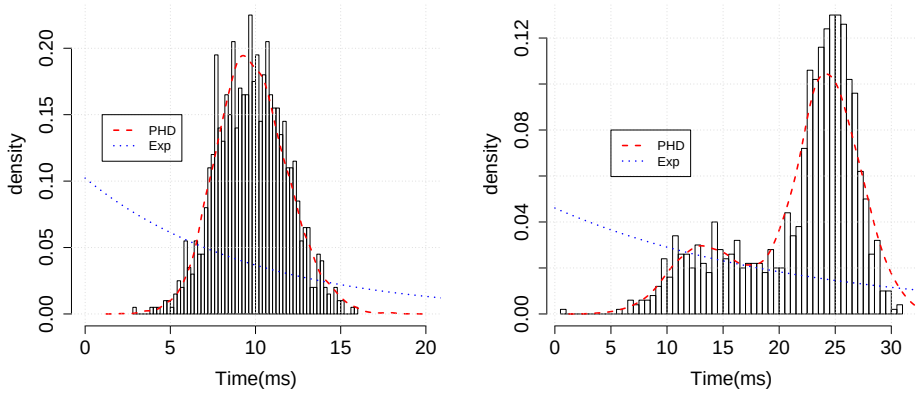
The OMNI HoldingTimeModeling function from Algorithm 1 achieves similar benefits by employing Reinecke et al.'s *cluster-based PHD fitting* approach [73, 83, 84] to fit a PHD to the holding time sample $\tau'_{i1}, \tau'_{i2}, \ldots, \tau'_{in_i}$ from (3.6). The PHD fitting is carried out by the while loop in lines 7–22, which iteratively assesses the suitability of PHDs obtained when partitioning the *sample* assembled in line 2 into $c = MinC, MinC + 1, \ldots, MaxC$ clusters. Line 8 obtains a PHD with c branches (corresponding to partitioning *sample* into c clusters) and up to *MaxP* phases by using the function CBFitting, which implements the cluster-based PHD fitting from [73]. 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.

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 HoldingTimeModeling(α, τ'_i1, τ'_i2, ..., τ'_in_i)
2:     sample ← (τ'_i1, τ'_i2, ..., τ'_in_i)
3:     minErr = ∞
4:     improvement ← 0
5:     steps ← 0
6:     c ← MinC
7:     while c ≤ MaxC ∧ steps ≤ MaxSteps do
8:         phd ← CBFitting(sample, c, FittingAlg, MaxP)
9:         err ← ΔCDF(sample, phd)
10:        if err < minErr then
11:            best_phd ← phd
12:            improvement ← improvement+(minErr−err)
13:            minErr ← err
14:        end if
15:        if improvement ≥ α then
16:            improvement ← 0
17:            steps ← 0
18:        else
19:            steps ← steps + 1
20:        end if
21:        c ← c + 1
22:    end while
23:    return best_phd
24: end function
```

The quality of the c-branch PHD is assessed in line 9 by using the CDF-difference metric [73] to compute the difference *err* between *sample* and the PHD. The if statement in lines 10–14 identifies the PHD with the lowest *err* value so far, retaining it in line 11. Reductions in *err* (i.e., "improvements") are cumulated in *improvement* (line 12), and the while loop terminates early if the iteration counter *steps* exceeds *MaxSteps* before *improvement* reaches the threshold $\alpha \geq 0$ provided as an parameter to HoldingTimeModeling and the *steps* counter is reset in line 17. Finally, the best PHD achieved within the while loop is returned in line 23.

Figure 3.10 shows examples of two PHDs generated by Algorithm 1 when applied to sample data produced by parametric distributions. Figure 3.10(a) illustrates a sample set of 1000 data points extracted from a normal distribution with a mean of 10ms and a standard deviation of 2ms, plotted as a histogram. Since an exponential model based on the first moment of the distribution is typically used to model holding times in a CTMC we also provide this as a comparison to the probability density estimation of the derived PHD. Figure 3.10(b) shows 1000 data points extracted from a bimodal distribution with one mode



(a) (b)

Figure 3.10: Distribution fitting: (a) Normal distribution; and (b) Bimodal Distribution

at 25ms and a second at 15ms along with the exponential estimator and a probability density estimation of the derived PHD. Visual inspection of the graphs show that in both cases the PHD is a much more accurate model of the sample set.

Using a phase-type distribution $\mathsf{PHD}(\boldsymbol{\pi}_0, \mathbf{D}_0)$ generated by Algorithm 1 to apply the OMNI holding-time modelling procedure to the state s' of a model $\mathsf{CTMC}(S', \boldsymbol{\pi}', \mathbf{R}')$ yields a model $\mathsf{CTMC}(S'', \boldsymbol{\pi}'', \mathbf{R}'')$ with:

$$S'' = (S' \setminus \{s'\}) \cup \{w_1, w_2, \ldots, w_N\} \ ;$$

$$\boldsymbol{\pi}''(s) = \begin{cases} \boldsymbol{\pi}'(s), & \text{if } s \in S \setminus \{s'\} \\ \boldsymbol{\pi}'(s')\boldsymbol{\pi}_0(w_i), & \text{if } s = w_i, 1 \leq i \leq N \\ 0, & \text{otherwise} \end{cases} ;$$

$$\mathbf{R}''(s,u) = \begin{cases} \mathbf{R}'(s,u), & \text{if } s,u \in S' \setminus \{s'\} \\ \mathbf{D}_0(s,u), & \text{if } s,u \in \{w_1, w_2, \ldots, w_N\} \\ \mathbf{R}'(s,s')\boldsymbol{\pi}_0(w_i), & \text{if } s \in S' \setminus \{s'\} \wedge u = w_i, \\ & \quad 1 \leq i \leq N \\ \frac{\mathbf{R}'(s',u)}{\lambda'}\mathbf{d}_1(w_i), & \text{if } s = w_i \wedge u \in S' \setminus \{s'\}, \\ & \quad 1 \leq i \leq N \end{cases}$$

if $s \neq u$; and $\mathbf{R}''(s,s) = -\sum_{u \in S'' \setminus \{s\}} \mathbf{R}''(s,u)$, where:

- $w_1, w_2, \ldots, w_N$ are the transient states of $\mathsf{PHD}(\boldsymbol{\pi}_0, \mathbf{D}_0)$;
- λ' is the total outgoing transition rate for s';
- $\mathbf{d_1} = -\mathbf{D}_0\mathbf{1}$.

Example 3.4. *We used our OMNI implementation to perform refinement of each web service in the motivating example. Algorithm 1 was executed for each of our six services, with* $\alpha = 0.1$ *and with the configuration parameters* $MinC = 2$, $MaxC = 30$, $MaxP = 300$, $MaxSteps = 3$ *and FittingAlg an expectation-maximisation PHD fitting algorithm that produces hyper-Erlang*

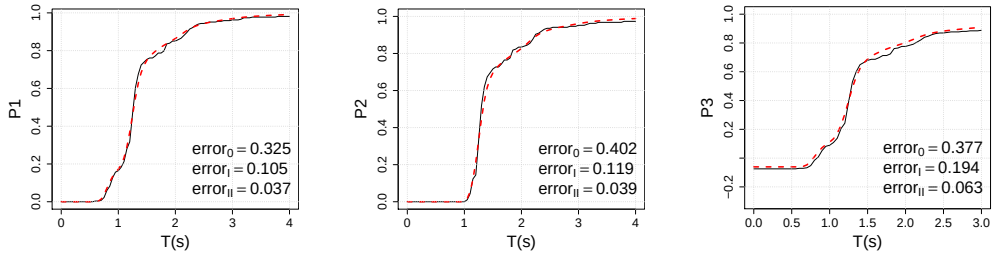


Figure 3.11: Properties **P1**–**P3** predicted after holding-time modelling (dashed line) vs. actual (continuous line); error_0, $\mathsf{error}_\mathsf{I}$ and $\mathsf{error}_\mathsf{II}$ are the prediction errors before OMNI and after each OMNI stage, respectively

*distributions. OMNI produced a refined CTMC with 1766 states and 1797 transitions. The time taken to verify properties for this larger model increased from less then 0.1 second, for the high-level CTMC, to approximately 5 seconds per property for the OMNI refined model. Figure 3.11 compares the actual values of properties **P1**–**P3** with the values predicted by the analysis of this CTMC. Both the visual assessment and the $\mathsf{error}_\mathsf{II}$ error values associated with these predictions (which are significantly lower than the error values error_0 before the OMNI refinement and $\mathsf{error}_\mathsf{I}$ after the first OMNI stage) show that this CTMC supports the accurate analysis of the three properties.*

3.3 Related work

Analytical analysis of non-Markovian processes is an active research area with phase-type Distributions being used to overcome limitations in the software tool support. OMNI builds on recent approaches to using PHDs to fit non-parametric distributions, a research area that has produced many efficient PHD fitting algorithms over the past decade [77],[81],[80],[82]. Buchholz et al. [39] and Okamura and Dohi [75] present overviews of the theory and applications of PHDs in these types of analysis, in domains including the modelling of call centres [41] and healthcare processes [40],[42]. However, these algorithms and applications consider the distribution of timing data for a complete end-to-end process rather than separate timing datasets for the activities within a larger system or process as is the case for OMNI. This focus on a single dataset also applies to the cluster-based PHD fitting method from [73] and its implementation within the efficient PHD-fitting tool HyperStar [83], which OMNI uses for its holding-time modelling.

Recent work by Karmakar and Gopinath [85] has shown that PHD models can be used in conjunction with CTMC solvers to verify storage reliability models. In this work, Weibull distributions are assumed to more accurately describe the processes of concern, and PHDs are used to approximate these distributions. The PRISM probabilistic model checker is then used to assess properties concerned with system reliability. Unlike this approach, OMNI is applicable to the much wider class of problems where additional QoS properties need to be analysed and where the relevant component features correspond to non-parametric distributions that cannot be accurately modelled as Weibull distributions.

The analysis of non-Markovian processes using PHDs is considered in [86], where a pro-

cess algebra is proposed for use with the probabilistic model checker PRISM. However, [86] 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 [44], where interval distributions are used to separate discrete and continuous features. Similar work [87] supports the synthesis of timeouts in fixed-delay CTMCs by using Markov decision processes. Unlike OMNI, [87],[44] 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, non-PHD-based approaches to combining Markov models with real data range from using Monte Carlo simulation to analyse properties of discrete-time Markov chains with uncertain parameters [88] to using semi-Markov chains to model holding times governed by general distributions [89]. 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.

Chapter 4

Property-Centric CTMC Refinement

In Chapter 3 we showed how, through the use of observation-enhanced Markov chain refinement the accuracy of transient QoS property analysis could be greatly improved. Whilst the refined CTMC models are more accurate they are also larger and more complex and therefore take more time to verify. Indeed we observed that applying the OMNI approach to a Markov model of a web application led to an increase in the number of states in the model from 8 to 1766.

In this Chapter we introduce a property-centric refinement method which, through the appropriate classification of system states, is able to maintain the accuracy of observation based Markov chain refinement whilst controlling the size of the resulting model. By analysing the original CTMC structure with respect to a QoS property of interest we are able to produce a smaller model for each property without losing model accuracy. Where multiple properties exist refinements are cached to reduce the construction time associated with other properties. Figure 4.1 shows an overview of the proposed refinement process.

The first OMNI step, called *Activity classification*, partitions the states of the high-level CTMC into subsets that require different types of refinement because of the different impact of their associated activities on the analysed property. For instance, activities which do not appear on an execution path have no effect on QoS properties (e.g. response time) associated solely with that path, and therefore their corresponding states from the high-level CTMC need not be refined.

The second OMNI step, called *selective refinement*, replaces the states which correspond to activities that impact the analysed property with new states and transitions that model the delays and holding times as outlined in Chapter 3.

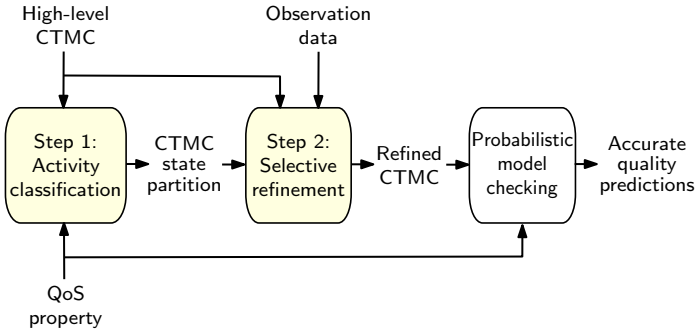


Figure 4.1: OMNI CTMC refinement and verification

4.1 Activity classification

CTMC models of system processes are typically constructed to allow for the verification of properties encoded in continuous stochastic logic (CSL) and to investigate questions about what might happen if the characteristics of activities were to be replaced. As such one model may contain information that is not required for the analysis of one or more CSL properties.

Given a high-level CTMC model $\mathsf{CTMC}(S, \boldsymbol{\pi}, \mathbf{R})$ of a process, and a QoS property encoded by the transient CSL formula $P_{=?}[\Phi_1 U^I \Phi_2]$, this OMNI step builds a partition

$$S = S_{\mathsf{X}} \cup S_{\mathsf{O}} \cup S_1 \cup S_2 \cup \ldots \cup S_m \tag{4.1}$$

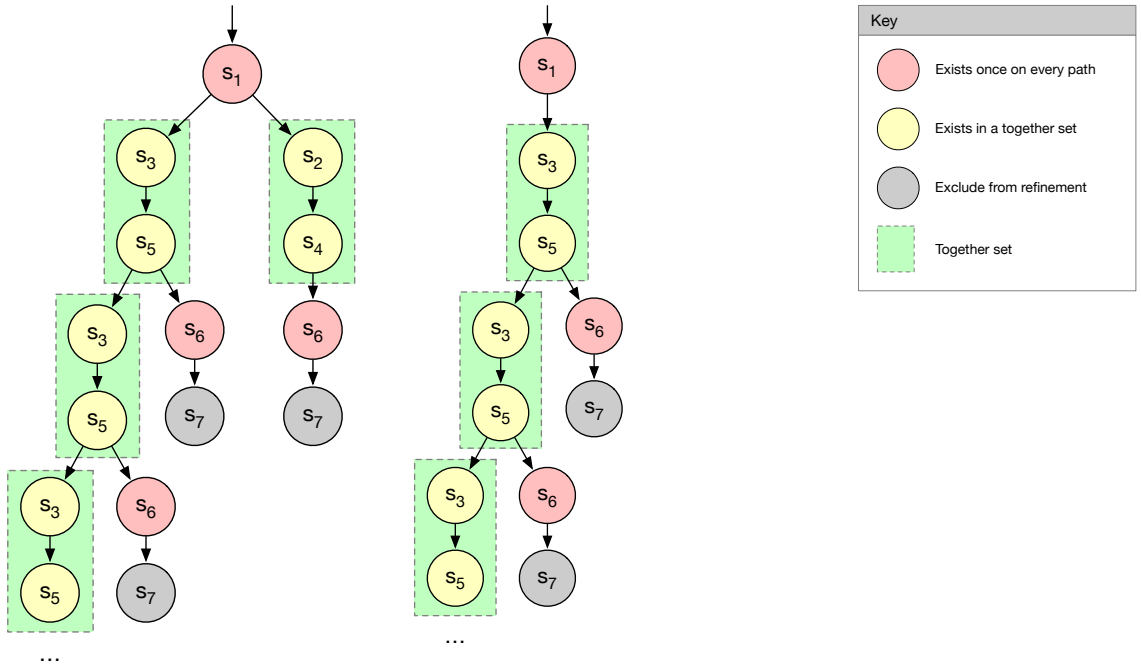
of the state set S. Intuitively, the "eXclude-from-refinement" set S_{X} will contain states with zero probability of occurring on paths that satisfy $\Phi_1 U^I \Phi_2$; the "Once-only" set S_{O} will contain states with probability 1.0 of appearing once and only once on every path that satisfies $\Phi_1 U^I \Phi_2$; and each "together" set S_i will contain states that can only appear as a sequence on paths that satisfy $\Phi_1 U^I \Phi_2$.

For simple Markov chain models the classification of states may be possible through a visual inspection. Let us consider the web based travel application used as a motivating example in Chapter 3 (Figure 3.2) and the properties **P1** and **P2** defined in Section 3.1.

Figure 4.2(a) illustrates all possible paths which satisfy property **P1** : $P_{=?}[F^{[0,T]} \mathit{complete}]$. Paths branch where a probabilistic choice exists in the CTMC and the total number of possible paths is infinite, however, all of the paths include the states s_1 and s_6 once and only once. These states therefore form the "Once-only" set. State s_7 denotes the end of the process and can therefore be "excluded" from the refinement. Finally we can see that where state s_3 exists on a path it is always immediately followed by state s_5, the same is true of states s_2 and s_4. These state pairs then form the "together" set.

Figure 4.2(b) shows the paths which satisfy property **P2** : $P_{=?}[\neg \mathit{arrivals}\ U^{[0,T]} \mathit{complete}]/(1-p_1)$. States s_2 and s_4 do not exist on any path which satisfies **P2** and hence are moved to the "eXclude-from-refinement" set.

More generally, however, analysis through visual inspection is not possible in this way and therefore we provide formal definitions of the disjoint sets S_{X}, S_{O}, and S_1 to S_m and descrip-



(a) Execution paths which satisfy property P1

(b) Execution paths which satisfy property P2

Figure 4.2: Activity classification

tions of their roles in OMNI in Sections 4.1.1–4.1.3.

4.1.1 Exclude-from-refinement state sets

Since activities which are not used can have no effect on the system performance the refinement of states associated with these activities is unnecessary and should be avoided. As such we wish to identify such states and so exclude them from the model refinement stage. For example, if we examine Figure 4.2b we observe that for P2 states s_2 and s_4 are not present on any path which does satisfy the QoS property. As such refining the states associated with these web services (arrivals, search) is not necessary.

Definition 4.1. *The* exclude-from-refinement state set S_X *associated with an until path formula* $P_{=?}[\Phi_1 U^I \Phi_2]$ *over the continuous-time Markov chain* $\mathcal{C} = \mathsf{CTMC}(S, \boldsymbol{\pi}, \mathbf{R})$ *is the set of CTMC states*

$$S_X = \{s \in S \mid P_{=?}[(\neg s \wedge \Phi_1) U \Phi_2] = P_{=?}[\Phi_1 U \Phi_2]\}, \tag{4.2}$$

where, for each state $s \in S$*, AP is extended with an atomic proposition also named 's' that is true in state* s *and false in every other state. Thus,* S_X *comprises all states* s *for which the probability* $P_{=?}[(\neg s \wedge \Phi_1) U \Phi_2]$ *of reaching a state satisfying* Φ_2 *along paths that* do not contain state s *and on which* Φ_1 *holds in all preceding states is the same as the probability* $P_{=?}[\Phi_1 U \Phi_2]$ *of reaching a state that satisfies* Φ_2 *along paths on which* Φ_1 *holds in all preceding states.*

Theorem 2. *Let S_{X} be the exclude-from-refinement state set associated with the until path formula $P_{=?}[\Phi_1 U^I \Phi_2]$ over the continuous-time Markov chain $\mathcal{C} = \mathsf{CTMC}(S, \boldsymbol{\pi}, \mathbf{R})$ with atomic proposition set AP. Then, for any $I \subseteq \mathbb{R}_{\geq 0}$, the probability $P_{=?}[\Phi_1 U^I \Phi_2]$ does not depend on the transition times from states in S_{X}.*

Proof. The proof is by contradiction. Consider a generic state $s_{\mathsf{X}} \in S_{\mathsf{X}}$ and the following sets of paths:

$$\begin{aligned}
A = \{\omega \in Paths^{\mathcal{C}} \mid \exists t > 0 \,.\, (\omega@t \models \Phi_2 \wedge \\
(\forall t' \in [0,t).\omega@t' \models \Phi_1))\} \\
B = \{\omega \in Paths^{\mathcal{C}} \mid \exists t > 0 \,.\, (\omega@t \models \Phi_2 \wedge \\
(\forall t' \in [0,t).\omega@t' \models \Phi_1 \wedge \omega@t' \neq s_X))\} \\
C = \{\omega \in Paths^{\mathcal{C}} \mid \exists t > 0 \,.\, (\omega@t \models \Phi_2 \wedge \\
(\forall t' \in [0,t).\omega@t' \models \Phi_1) \wedge (\exists t' \in [0,t).\omega@t' = s_{\mathsf{X}}))\}
\end{aligned}$$

As $A = B \cup C$ and $B \cap C = \emptyset$, we have $Pr_{\boldsymbol{\pi}}(A) = Pr_{\boldsymbol{\pi}}(B) + Pr_{\boldsymbol{\pi}}(C)$. However, according to (4.2), $Pr_{\boldsymbol{\pi}}(A) = P_{=?}[\Phi_1 U \Phi_2] = P_{=?}[(\neg s \wedge \Phi_1) U \Phi_2] = Pr_{\boldsymbol{\pi}}(B)$, so $Pr_{\boldsymbol{\pi}}(C) = 0$.

Assume now that the time spent by the CTMC in state s_{X} has an impact on the value of $P_{=?}[\Phi_1 U^I \Phi_2]$ over $Paths^{\mathcal{C}}$ for an interval $I \subseteq \mathbb{R}_{\geq 0}$. This requires that, at least for some (possibly very small) values of the time $t_X > 0$ spent in s_{X}, s_{X} appears on paths from a set

$$\begin{aligned}
C' = \{\omega \in Paths^{\mathcal{C}} \mid \exists t \in I \,.\, (\omega@t \models \Phi_2 \wedge \\
(\forall t' \in [0,t).\omega@t' \models \Phi_1) \wedge (\exists t' \in [0,t).\omega@t' = s_{\mathsf{X}}))\}
\end{aligned}$$

such that $Pr_{\boldsymbol{\pi}}(C') > 0$; otherwise, varying t_X cannot have any impact on

$$\begin{aligned}
P_{=?}[\Phi_1 U^I \Phi_2] = Pr_{\boldsymbol{\pi}}\{\omega \in Paths^{\mathcal{C}} \mid \exists t \in I \,.\, (\omega@t \models \Phi_2 \wedge \\
(\forall t' \in [0,t).\omega@t' \models \Phi_1))\}
\end{aligned}$$

However, since $C' \subseteq C$ we must have $Pr_{\boldsymbol{\pi}}(C) \geq Pr_{\boldsymbol{\pi}}(C') > 0$, which contradicts our earlier finding that $Pr_{\boldsymbol{\pi}}(C) = 0$, completing the proof. □

Theorem 2 allows OMNI to leave the states from S_{X} unrefined with no loss of accuracy in the QoS analysis results. The theorem also provides a method for obtaining S_{X} by computing the until formula $P_{=?}[(\neg s \wedge \Phi_1) U \Phi_2]$ for each state s of the high-level CTMC (i.e. for each activity) and comparing the result with the value of the CSL formula $P_{=?}[\Phi_1 U \Phi_2]$, which is only computed once. Existing probabilistic model checkers compute these *unbounded* until formulae very efficiently, as they only depend on the probabilities of transition between CTMC states and not on the state transition rates [27],[38].[1]

[1] To asses the time taken by model checking, an experiment was carried out to evaluate each state from the motivating example for inclusion in S_{X}. This experiment was repeated 30 times and the average time taken by model checking each state was found to be 1.6ms.

Example 4.1. *Consider the QoS properties (3.2) of the web application from our motivating example. For property* ***P2*** *and the high-level CTMC model from Figure 3.2, we have*

$$\begin{aligned} P_{=?}[\neg \mathit{arrivals}\ U\ \mathit{complete}] = 1 - p_1 = \\ P_{=?}[(\neg s_2 \wedge \neg \mathit{arrivals})\ U\ \mathit{complete}] = \\ P_{=?}[(\neg s_4 \wedge \neg \mathit{arrivals})\ U\ \mathit{complete}] = \\ P_{=?}[(\neg s_7 \wedge \neg \mathit{arrivals})\ U\ \mathit{complete}], \end{aligned}$$

(and $P_{=?}[(\neg s \wedge \neg \mathit{arrivals})\, U \mathit{complete}] \neq 1-p_1$ *for any other state* s*), so* $S_{?} = \{s_2, s_4, s_7\}$ *for* ***P2****. Applying Theorem 2 to the other two properties from (3.2) yields* $S_{?} = \{s_7\}$*.*

4.1.2 Once-only state sets

In this section we will show that for any states which occur once-only on all paths which satisfy the QoS property it is not necessary to model the delay associated with the activity which the state represents. Since delays are expensive to model this can reduce the size and complexity of the resulting model.

Let us consider the CTMC model shown in Figure 4.3a where each state s_i has a deterministic delay d_i. Figure 4.3b then shows the model after the each state has been refined such that each state s_i is replaced by an Erlang distribution representing the delay d_i and an phase-type distribution PHD_i which models the residual holding time.

For a property $P_{=?}[F^T \mathit{complete}]$ we are only concerned in the time taken to reach the "complete" state and hence the order in which activities are carried out is not important as long as we maintain the temporal characteristics of the model. The activities represented by states s_1 and s_4 occur once, and only once, for all paths which satisfy this property. We know, therefore, that the delays associated with those states are also experienced once for every process execution that satisfies the property of interest. We can, therefore, re-arrange the chain as shown in Figure 4.3c, moving the delays to the start of the chain and summing them to form a single delay.

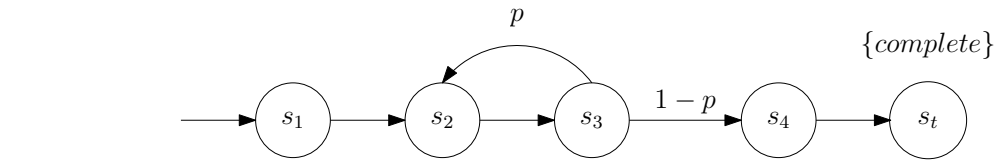
The probability of reaching *complete* in the time interval $[0, d_1 + d_4)$ is then 0 and model checking is not required in this interval. Instead we may remove the delay as shown in Figure 4.3d and return a probability of zero for $T < (d_1 + d_4)$.

Definition 4.2. *The* once-only state set $S_{?}$ *associated with an until path formula* $P_{=?}[\Phi_1 U^I \Phi_2]$ *over the continuous-time Markov chain* $\mathcal{C} = \mathsf{CTMC}(S, \boldsymbol{\pi}, \mathbf{R})$ *is the set*

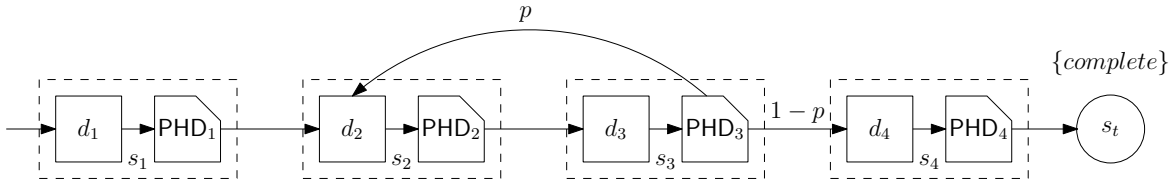
$$\begin{aligned} S_{?} = \{ s \in S \setminus S_{?} \mid P_{>0}[\Phi_1 U \Phi_2] \wedge P_{\leq 0}[(\neg s \wedge \Phi_1) U \Phi_2] \wedge \\ \forall s' \in S\,.\,(s \models P_{>0}[X s'] \rightarrow s' \models P_{\leq 0}[\neg S_{?} U s]) \}, \end{aligned} \tag{4.3}$$

where the until formula $\neg S_{?} U s$ *holds for paths that reach state* s *without going through any states from* $S_{?}$ *(which corresponds to labelling the states from* $S_{?}$ *with the atomic proposition '*$S_{?}$*').*

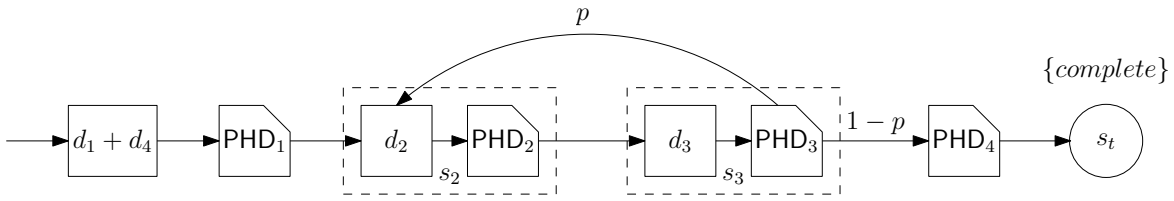
The next theorem asserts that for every state $s_{?}$ from $S_{?}$, $P_{=?}[\Phi_1 U^I \Phi_2]$ can be calculated by applying the probability measure $Pr_{\boldsymbol{\pi}}$ to the set of paths ω which, in addition to satisfying



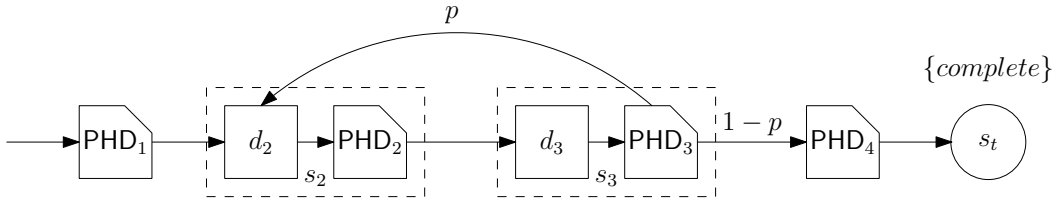
(a) Original CTMC model



(b) OMNI model with delays modelled as Erlang distributions and holding times modelled with PHDs



(c) Combining once only delays



(d) Removing once only delays

Figure 4.3: Removing delays from once-only states

the clause specified by the CSL semantics (i.e., $\exists t \in I.(\forall t' \in [0,t).\, \omega@t' \models \Phi_1) \wedge \omega@t \models \Phi_2$), contain s_{O} once and only once *before time instant* t. Using the unique existential quantifier $\exists!$, the last clause can be formalised as $\exists! i\, .\, (\omega[i] = s_{\mathsf{O}} \wedge \sum_{j=1}^{i} t_j < t)$, where t_j is the time spent in the j-th state on the path.

Theorem 3. *Let S_{O} be the once-only state set associated with the until path formula $P_{=?}[\Phi_1 U^I \Phi_2]$ over the continuous-time Markov chain $\mathcal{C} = \mathsf{CTMC}(S, \boldsymbol{\pi}, \mathbf{R})$. Then, for any state $s_{\mathsf{O}} \in S_{\mathsf{O}}$ and interval $I \subseteq \mathbb{R}_{\geq 0}$,*

$$\begin{aligned} P_{=?}[\Phi_1 U^I \Phi_2] = Pr_{\boldsymbol{\pi}}\{\omega \in Paths^{\mathcal{C}} \mid \\ \exists t \in I\, .\, (\omega@t \models \Phi_2 \wedge (\forall t' \in [0,t)\, .\, \omega@t' \models \Phi_1) \wedge \\ \exists! i\, .\, (\omega[i] = s_{\mathsf{O}} \wedge \textstyle\sum_{j=1}^{i} t_j < t))\}. \end{aligned} \tag{4.4}$$

Proof. Let A' denote the subset of $Paths^{\mathcal{C}}$ from (4.4). According to CSL semantics, $P_{=?}[\Phi_1 U^I \Phi_2] = Pr_{\boldsymbol{\pi}}(A)$ where

$$\begin{aligned} A = \{\omega \in Paths^{\mathcal{C}} \mid \exists t \in I\, .\, (\omega@t \models \Phi_2 \wedge \\ (\forall t' \in [0,t).\omega@t' \models \Phi_1))\}. \end{aligned}$$

Since $A' \subseteq A$, we have $P_{=?}[\Phi_1 U^I \Phi_2] = Pr_{\boldsymbol{\pi}}(A) = Pr_{\boldsymbol{\pi}}(A') + Pr_{\boldsymbol{\pi}}(A \setminus A')$, so to prove the theorem we must show that $Pr_{\boldsymbol{\pi}}(A \setminus A') = 0$. To this end, we partition $A \setminus A'$ into two disjoint subsets: A_1, comprising the paths that do not contain state s_{O} before time t from the first line of (4.4), and A_2, comprising the paths that contain state s_{O} before time t more than once. Since $P_{\leq 0}[(\neg s_{\mathsf{O}} \wedge \Phi_1) U \Phi_2]$ holds (according to the definition of S_{O}), $Pr_{\boldsymbol{\pi}}(A_1) = 0$. Similarly, since $\forall s' \in S \,.\, (s_{\mathsf{O}} \models P_{>0}[X s'] \rightarrow s' \models P_{\leq 0}[\neg S_{\mathsf{X}} U s_{\mathsf{O}}])$ holds, the set of paths satisfying $\Phi_1 U^I \Phi_2$ and containing s_{O} twice (without reaching states in S_{X}) occur with probability zero. As A_2 is included in this set, we necessarily have $Pr_{\boldsymbol{\pi}}(A_2) = 0$. We conclude that $Pr_{\boldsymbol{\pi}}(A \setminus A') = Pr_{\boldsymbol{\pi}}(A_1) + Pr_{\boldsymbol{\pi}}(A_2) = 0$, which completes the proof. □

OMNI exploits Theorem 3 in two ways. First, since S_{O} states correspond to activities always executed before $\Phi_1 U^I \Phi_2$ becomes true, $P_{=?}[\Phi_1 U^I \Phi_2] = 0$ for any interval $I \subseteq \left[0, \sum_{s_i \in S_{\mathsf{O}}} \delta_i\right)$, where δ_i is the delay (3.5) of the activity i associated with state s_i. Therefore, OMNI returns a zero probability in this scenario without performing probabilistic model checking. Second, because the activities associated with S_{O} states are executed precisely once on relevant CTMC paths, no modelling of their delays is required, and OMNI only needs to model the holding times of these states. Importantly, obtaining S_{O} to enable these simplifications only requires the probabilities of unbounded until and next path formulae (cf. (4.3)), which probabilistic model checkers can compute efficiently for the reasons we explained earlier in this section.

Example 4.2. *Consider property **P1** from the QoS properties (3.2) in our motivating example: $P_{=?}[F^{[0,T]}\, complete]$. In line with definition (4.3), we obtain the set S_{O} for this property by first evaluating the following CSL formulae for the high-level CTMC from Figure 3.2:*

- $P_{>0}[true\, U\, complete]$ *which holds as* $P_{=?}[F\ complete] = 1$
- $P_{\leq 0}[(\neg s \wedge true)\, U\, complete] = P_{\leq 0}[\neg s\, U\, complete]$, *which holds only for states* s_1 *and* s_6.

The constraint $\forall s' \in S.(s \models P_{>0}[X s'] \rightarrow s' \models P_{\leq 0}[\neg S_{\mathsf{X}} U s])$ is then checked only for the S_{O}-candidate states $s = s_1$ and $s = s_6$, taking into account the fact that $S_{\mathsf{X}} = \emptyset$ (cf. Example 4.1). For instance, since $s_1 \models P_{>0}[X s']$ only for $s' \in \{s_2, s_3\}$, and $s_2, s_3 \models P_{\leq 0}[true\ U\ s_1]$, we conclude that $s_1 \in S_{\mathsf{O}}$. Similarly, $s_6 \models P_{>0}[X s']$ only if $s' = s_7$ and $s_7 \models P_{\leq 0}[true\ U\ s_6]$, so $s_6 \in S_{\mathsf{O}}$, giving $S_{\mathsf{O}} = \{s_1, s_6\}$. It is easy to show that the same "once-only" state set is obtained for the other two properties from (3.2).

4.1.3 Together state sets

Finally, the result in this section supports the calculation and exploitation of the "together" state sets from (4.1). If we consider once more the CTMC fragment as shown in Figure 4.3a then we observe that on any path s_2 is always followed by s_3 and, s_3 is always preceded by state s_2. For this sequence of states we may then combine the delays associated with each activity and reduce the number of delays to be modelled in the system. The result of combining these delays is shown in Figure 4.4. Comparing this model with the initial CTMC fragment as refined by OMNI without property-centric refinement (Figure 4.3b) shows that we have reduced the number of delays which require modelling from 3 to 1.

Algorithm 2 Generation of "together" state sequences

```
1: function TogetherSeqs(CTMC(S, π, R), S_X, S_O)
2:     TS ← ∅, States ← S \ (S_X ∪ S_O)
3:     while States ≠ ∅ do
4:         s ← PickAnyElement(States)
5:         T ← ⟨s⟩, States ← States \ {s}
6:         left, right ← true
7:         while (left ∨ right) ∧ States ≠ ∅ do
8:             if left then
9:                 s ← Pred(Head(T), States, S, π, R)
10:                if s ≠ NIL then
11:                    T ← ⟨s⟩⌢T, States ← States \ {s}
12:                else
13:                    left ← false
14:                end if
15:            end if
16:            if right then
17:                s ← Succ(Tail(T), States, S, π, R)
18:                if s ≠ NIL then
19:                    T ← T⌢⟨s⟩, States ← States \ {s}
20:                else
21:                    right ← false
22:                end if
23:            end if
24:        end while
25:        TS ← TS ∪ {T}
26:    end while
27:    return TS
28: end function

29: function Pred(s, States, S, π, R)
30:    if π(s) > 0 then return NIL end if
31:    for s′ ∈ States do
32:        if R(s′, s) > 0 ∧ ∀s″ ∈ S \ {s, s′} .
                    (R(s′, s″) = 0 ∧ R(s″, s) = 0) then
33:            return s′
34:        end if
35:    end for
36:    return NIL
37: end function

38: function Succ(s, States, S, π, R)
39:    for s′ ∈ States do
40:        if π(s′) = 0 ∧ R(s, s′) > 0 ∧ ∀s″ ∈ S \ {s, s′} .
                    (R(s, s″) = 0 ∧ R(s″, s′) = 0) then
41:            return s′
42:        end if
43:    end for
44:    return NIL
45: end function
```

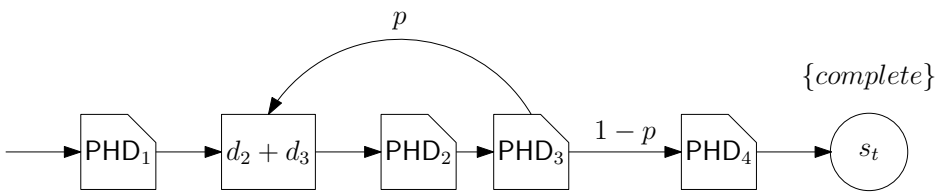


Figure 4.4: CTMC model showing the combination of delays from together states

The function TogetherSeqs, from Algorithm 2, takes as input the $\mathsf{CTMC}(S, \boldsymbol{\pi}, \mathbf{R})$ and the state sets S_{X} and S_{O} for the QoS property $P_{=?}[\Phi_1 U \Phi_2]$, and builds the m state sequences in successive iterations of its outer while loop (lines 3–26). The set *States* maintains the states yet to be allocated to sequences (initially $S \setminus (S_{\mathsf{X}} \cup S_{\mathsf{O}})$, cf. line 2), and each new sequence T starts with a single element picked randomly from *States* (line 4). The inner while loop in lines 7–24 "grows" this sequence. First, the if statement in lines 8–15 tries to grow the sequence to the left with a state s that "precedes" the sequence, in the sense that the only outgoing CTMC transition from s is to the sequence head, and the only way of reaching the sequence head is through an incoming CTMC transition from s. Analogously, the if statement in lines 16-23 grows the sequence to the right, by appending to it the state that "succeeds" the state at the tail of the sequence, if such a "successor" state exists. The predecessor and successor states of a state s are computed by the functions Pred and Succ, respectively, where these functions return NIL if the states they attempt to find do not exist. The inner while loop terminates when the *States* set becomes empty or the sequence T has no more predecessors or successors, so the flags *left* and *right* are set to *false* in lines 13 and 21, respectively. On exit from this while loop, the sequence T is added to the set of sequences TS, which is returned (line 27) after the outer while loop also terminates when *States* becomes empty. Termination is guaranteed since at least one element is removed from *States* in each iteration of this while loop (in line 5).

To analyse the complexity of TogetherSeqs, we note that the worst case scenario corresponds to $S_{\mathsf{X}} = S_{\mathsf{O}} = \emptyset$ and to the function returning only sequences of length 1, in which case the outer while loop is executed $|S|$ times with both Pred and Succ invoked once in each iteration. The if statements from Pred and Succ perform $\mathsf{O}(|S|)$ comparisons, and are executed within for loops with $\mathsf{O}(|S|)$ iterations, yielding an $\mathsf{O}(|S|^2)$ complexity for each function, and an overall $\mathsf{O}(|S|^3)$ complexity for the algorithm. Calculation of together states is therefore practical with modest computing hardware since the number of states in the high-level CTMC is a small, since a single state is used to represent each task or activity in the operational process.

Theorem 4. *If $T = \langle s_{i1}, s_{i2}, \ldots, s_{iN_i} \rangle$ is one of the sequences returned by* TogetherSeqs*, ω a path that satisfies $\Phi_1 U^I \Phi_2$ for an interval $I \subseteq \mathbb{R}_{\geq 0}$, and $t \in I$ the earliest time when $\omega@t \models \Phi_2$ (with $\omega@t' \models \Phi_1$ for all $t' \in [0, t)$), then up to time t the states from T can only appear on ω as complete sequences $\ldots s_{i1} t_{i1} s_{i2} t_{i2} \ldots s_{iN_i} t_{iN_i} \ldots$.*

Proof. The case $N_i = 1$ is trivial, so we assume $N_i > 1$ in the rest of the proof. We have two cases: either ω contains no states from T, or it contains at least one state from T. In the former case, the theorem is proven. In the latter case, consider any state s_{i_j} that occurs on ω, $1 \leq j \leq N_i$. The states s_{i1}, s_{i2}, …, $s_{i,j-1}$ must also occur on ω, in this order and just before s_{ij}, as transitioning through each of these states is the only way to reach s_{ij} in the

Table 4.1: CTMC state partition for the web application properties

Property	S_{X}	S_{O}	$S_1, S_2, \ldots, S_m$
P1	$\{s_7\}$	$\{s_1, s_6\}$	$\{s_2, s_4\}, \{s_3, s_5\}$
P2	$\{s_2, s_4, s_7\}$	$\{s_1, s_6\}$	$\{s_3, s_5\}$
P3	$\{s_7\}$	$\{s_1, s_6\}$	$\{s_2, s_4\}, \{s_3, s_5\}$

CTMC. Moreover, $s_{i,j+1}$, $s_{i,j+2}$, …, s_{iN_i} must immediately follow s_{ij} on ω (in this order) because s_{ij} is not an absorbing state and its only outgoing transition is to $s_{i,j+1}$, etc. Hence, the path is of the form $\omega = s_1 t_1 s_2 t_2 \ldots s_x t_x s_{i1} t_{i1} s_{i2} t_{i2} \ldots s_{iN_i} t_{iN_i} \ldots$ for some $x \geq 0$. To prove that this occurrence of all states from T on ω is either up to or after time t, we show that it is not possible to have $\omega@t = s_{ij}$ for any $j < N$. Indeed, if we assume $\omega@t = s_{ij}$ then according to the hypothesis $s_{ij} \models \Phi_2$ must hold. As this must be true not only for ω but also for any other path ω' that satisfies $\Phi_1 U \Phi_2$ and contains the states from T, definition (4.2) implies that $s_{i,j+1}, s_{i,j+2}, \ldots, s_{iN_i} \in S_{\mathsf{X}}$ because ω' comprises states that satisfy Φ_1 followed by state s_{ij} that satisfies Φ_2, followed by the $s_{i,j+1}, s_{i,j+2}, \ldots, s_{iN_i}$. However, having states from T in S_{X} is not possible since line 2 of TogetherSeqs removes S_{x} from the set of states used to generate T. □

Theorem 4 allows OMNI to model the delays of all states in the same "together" set S_i, $1 \leq i \leq m$, as a *joint delay*

$$\Delta_i = \sum_{j=1}^{N_i} \delta_{ij}, \tag{4.5}$$

since the relevant part of any path that influences the value of $P_{=?}[\Phi_1 U^I \Phi_2] = 0$ contains either all these states or none of them.

Example 4.3. *Consider again the QoS properties (3.2) of the web application from our motivating example. For property **P2**,* TogetherSeqs *is called with* $S_{\mathsf{X}} = \{s_2, s_4, s_7\}$ *(cf. Example 4.1) and* $S_{\mathsf{O}} = \{s_1, s_6\}$ *(cf. Example 4.2), so it starts with* $States = \{s_1, s_2, \ldots, s_7\} \setminus (S_{\mathsf{X}} \cup S_{\mathsf{O}}) = \{s_3, s_5\}$ *in line 2. Irrespective of which of* s_3 *and* s_5 *is picked in line 5, the other state will be added to the same "together" sequence since the two states always follow one another with no intermediate states. The "together" sets for the other two properties from (3.2) are given in Table 4.1, which brings together the results from Examples 4.1–4.3.*

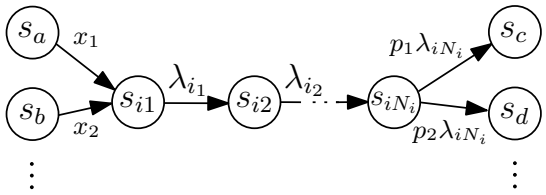
4.2 Selective refinement

The second OMNI step models the delays and holding times of the relevant activities of the analysed system according to the rules established in the previous section. These rules are summarised in Table 4.2, and require methods for joint delay modelling (for activities associated with "together" CTMC states) and for individual holding time modelling (for activities not associated with S_{X} states). The two methods are described next.

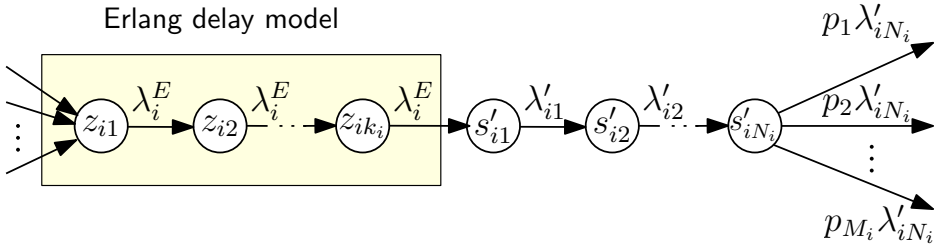
Table 4.2: OMNI rules for modelling the delays and holding times of different types of process activities

	S_{X} **activities**	S_{O} **activities**	S_i **activities**, $1 \leq i \leq m$
delay	no modelling needed	no modelling needed†	joint delay modelling
holding time	no modelling needed	per activity modelling	per activity modelling

$^{\dagger}S_{\mathsf{O}}$ activities introduce a deterministic delay $\sum_{s_i \in S_{\mathsf{O}}} \delta_i$



(a) Model of "together" states $i1, i2, \ldots, iN_i$ in the high-level CTMC (showing generic incoming transitions of rates $x_1, x_2, \ldots$ from states $s_a, s_b, \ldots$ into state s_{i1}, and generic outgoing transitions of probabilities $p_1, p_2, \ldots$ and rate λ_{iN_i} from state s_{iN_i} to states $s_c, s_d, \ldots$)



(b) Model of activities $i1, i2, \ldots, iN_i$ after joint delay modelling

Figure 4.5: Joint delay modelling for "together" state set S_i whose final state s_{iN_i} has $M_i \geq 1$ outgoing transitions

4.2.1 Joint delay modelling

For each "together" set $S_i = \{s_{i1}, s_{i2}, \ldots, s_{iN_i}\}$, OMNI extends the CTMC with additional states and transitions that model the joint delay Δ_i from (4.5) by means of an Erlang distribution as described in section 3.2.2. As shown in Figure 4.5 (c.f. Figure 3.7), this involves replacing the states from S_i with a sequence of delay-modelling states z_{i1}, z_{i2}, …, z_{ik_i} that encode an Erlang-k_i distribution of rate λ_i^E, followed by states $s'_{i1}, s'_{i2}, \ldots, s'_{iN_i}$ with transitions matching those from the high-level CTMC but of rates λ'_{i1}, λ'_{i2}, …, λ'_{iN}.

Theorem 1 supports the calculation of the delay model parameters for a "together" state set $S_i = \{s_{i1}, s_{i2}, \ldots, s_{iN_i}\}$ as follows:

1. Approximate the delays $\delta_{i1}, \delta_{i2}, \ldots, \delta_{iN_i}$ for the activities associated with each state from S_i using (3.5).
2. Compute the joint delay $\Delta_i = \sum_{j=1}^{N_i} \delta_{ij}$.
3. Choose a small error $\epsilon \in (0, 1)$ and a small probability p (e.g. $\epsilon = 0.1$ and $p = 0.05$), and

solve (3.7a) for k_i;

4. Calculate λ_i^E and λ'_{i1} to λ'_{iN_i} using (3.7b) and (3.7c), respectively.

Example 4.4. *Consider the "together" set $S_1 = \{s_2, s_4\}$ for property **P1** from our running example (cf. Table 4.1). States s_2 and s_4 correspond to the invocations of the arrivals and search web services from the travel web application, which according to our experimental data have delays $\delta_2 = 45ms$ and $\delta_4 = 209ms$, respectively. Therefore, the joint delay is $\Delta_1 = \delta_2 + \delta_4 = 254ms$. Suppose that we want to model this joint delay with an error bound $\epsilon = 0.1$ and a probability $p = 0.05$. This gives $k_1 = 259$ (cf. Table 3.2) and the other joint delay modelling parameters are calculated as: $\lambda_1^E = \frac{k_1}{\Delta_1} = \frac{259}{0.254} = 1019s^{-1}$, $\lambda'_2 = \frac{\lambda_2}{1-\lambda_2\delta_2} = \frac{19.88}{1-19.88\cdot 0.045} = 188.61s^{-1}$ and $\lambda'_4 = \frac{\lambda_4}{1-\lambda_4\delta_4} = \frac{1.85}{1-1.85\cdot 0.209} = 3.01s^{-1}$ (where the rates λ_2 and λ_4 are taken from Table 3.1).*

The next theorem gives the format of the refined CTMC after joint delay modelling is applied to all "together" state sets S_1 to S_m.

Theorem 5. *Applying the OMNI joint delay modelling procedure to the "together" state set S_i of a high-level model $\mathsf{CTMC}(S, \boldsymbol{\pi}, \mathbf{R})$ yields a model $\mathsf{CTMC}(S', \boldsymbol{\pi}', \mathbf{R}')$ with:*

$$S' = (S \setminus S_i) \cup \{z_{i1}, z_{i2}, \ldots, z_{ik_i}, s'_{i1}, s'_{i2}, \ldots, s'_{iN_i}\};$$

$$\boldsymbol{\pi}'(s) = \begin{cases} \boldsymbol{\pi}(s), & \text{if } s \in S \setminus S_i \\ \boldsymbol{\pi}(s_{i1}), & \text{if } s = z_{i1} \\ 0, & \text{otherwise} \end{cases};$$

$$\mathbf{R}'(s,u) = \begin{cases} \mathbf{R}(s,u), & \text{if } s,u \in S \setminus S_i \\ \mathbf{R}(s,s_{i1}), & \text{if } s \in S \setminus S_i \wedge u = z_{i1} \\ \lambda_i^E, & \text{if } (s,u) \in \{(z_{i1},z_{i2}), \ldots, \\ & \quad (z_{i,k_i-1}, z_{ik_i}), (z_{ik_i}, s'_{i1})\} \\ \lambda'_{ij}, & \text{if } s = s'_{ij} \wedge u = s'_{i,j+1}, \\ & \quad 1 \le j \le N_i - 1 \\ \frac{\mathbf{R}(s_{iN_i},u)}{\lambda_{iN_i}}\lambda'_{iN_i}, & \text{if } s = s'_{iN_i} \wedge u \in S \setminus S_i \\ \frac{\mathbf{R}(s_{iN_i},s_{i1})}{\lambda_{iN_i}}\lambda'_{iN_i}, & \text{if } s = s'_{iN_i} \wedge u = z_{i1} \\ 0, & \text{otherwise} \end{cases}$$

if $s \neq u$; and $\mathbf{R}'(s,s) = -\sum_{u \in S' \setminus \{s\}} \mathbf{R}'(s,u)$, where the terms $\frac{\mathbf{R}(s_{iN_i},u)}{\lambda_{iN_i}}$ and $\frac{\mathbf{R}(s_{iN_i},s_{i1})}{\lambda_{iN_i}}$ correspond to the probabilities $p_1, p_2, \ldots$ from Figure 4.5 and are obtained using (2.6).

Proof. The proof is by construction, cf. Figure 4.5. □

4.2.2 Holding-time modelling

As indicated in Table 4.2, we model the holding times of process activities associated with high-level CTMC states from $S_0 \cup S_1 \cup S_2 \ldots \cup S_m$ individually. For each such activity, we

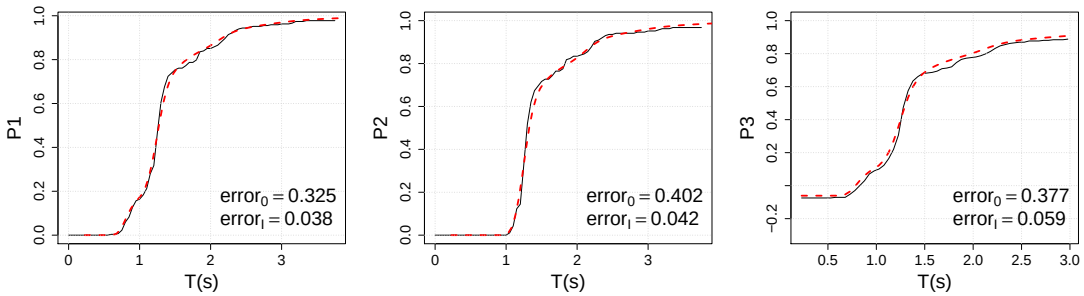


Figure 4.6: Actual (continuous lines) property values versus property values predicted (dashed lines) using the OMNI-refined CTMC model; error_0 and $\mathsf{error}_\mathsf{I}$ represent the error (3.4) for the high-level CTMC and the refined CTMC, respectively.

synthesise a phase-type distribution $\mathsf{PHD}(\boldsymbol{\pi}_0, \mathbf{D}_0)$ that models the holding times (3.6), and we replace the relevant state s' of the model $\mathsf{CTMC}(S', \boldsymbol{\pi}', \mathbf{R}')$ obtained after the OMNI joint delay modelling with this PHD. For operations corresponding to states $s_\mathsf{O} \in S_\mathsf{O}$ the replaced state is $s' = s_\mathsf{O}$, while for operations corresponding to a state $s_{ij} \in S_i$, $1 \leq i \leq m, 1 \leq j \leq N_i$, the replaced state is the state $s' = s'_{ij}$ obtained after the joint delay modelling of S_i (cf. Figure 4.5b).

Example 4.5. *We used our OMNI refinement tool (Section 5.1) to perform the CTMC state classification and selective refinement steps of our approach on the high-level CTMC from the motivating example. Algorithm 1 was executed for each activity associated with a CTMC state from the S_O or the S_1 to S_m state sets in Table 4.1, with $\alpha = 0.1$ and with the configuration parameters $MinC = 2$, $MaxC = 30$, $MaxP = 300$, $MaxSteps = 3$ and FittingAlg an expectation-maximisation PHD fitting algorithm that produces hyper-Erlang distributions. We obtained refined CTMCs comprising 730 states and 761 transitions for property **P1**, 367 states and 387 transitions for property **P2**, and 730 states and 761 transitions for property **P3**. Each of the models produced are therefore considerably smaller than the 1766 state model derived without property-centric refinement. Figure 4.6 compares the actual values of properties **P1**–**P3** with the values predicted by the analysis of these refined CTMCs.*

4.3 Related work

Whilst there has been major advances in model checking over recent years [55] one problem that remains is that of a "State space explosion" [90] where the number of states required to represent complex real-world systems results in models which are too large for modern model checking engines.

This challenge has long been an area of research with Clarke and Emerson using synchronization skeletons [24, 91] as an abstraction of concurrent programs. In such models, any detail irrelevant to the synchronization is suppressed and the complexity of temporal logic model checking reduced [92]. This method of deriving models requires a deep understanding of the system under investigation however. To tackle this Counterexample-Guided Abstraction Refinement (CEGAR) [93],[90] was proposed and abstraction was describes as the "most

important technique for handling this (state explosion) problem". In [94] a probabilistic version of CEGAR was developed in which the authors stated that the major challenge associated with extending CEGAR to probabilistic systems was the notion and analysis of counterexamples. CEGAR uses an "upper approximation" of the original system such that when a specification is true in the abstract model it is also true in the concrete design. Failure to meet specification may, however, be due to artefacts in the abstraction and so the abstraction is refined until the behaviour is eliminated. State explosion is reduced by only refining those states in the counterexample trace.

Whilst OMNI uses the concept of abstraction refinement the approach used is quite different from that proposed in earlier works. OMNI makes use of the high-level CTMC to reason about the temporal characteristics of activities . The refinement then introduces new states to the CTMC that did not exist in the original model. These new states do not change model structure but instead increase the accuracy of the temporal modelling in line with observed traces.

A second method for controlling the state space explosion problem is to use compositional verification techniques [95] which are applicable to component based systems such as those considered by OMNI. Assume-Guarantee reasoning is one such approach and was first advocated in the context of temporal logic by Pnuelli [96] and has since been shown to be effective in extending the applicability of formal verification to large component based systems [97],[98],[99],[100],[101]. Compositional verification techniques analyse system components independently and derive global system properties through a composition of the component-level properties and proof rules. The construction of abstract models for analysis in OMNI is different. The model which represents the structure of the process is defined with only one state per activity. Activities are therefore decoupled as part of the modelling process and the effect of one activity on another not need to be reasoned about in the analysis phase.

Recent work by Ashok et al. [102] proposed a method for speeding up reachability analysis for continuous-time Markov Decision processes that, like OMNI, only explores a partial model. The approach involves adding states to the system for analysis until the change in the reachability property is small enough that the process terminates. Unlike OMNI however the analysis starts with a subset of states from the complete CTMC model and through simulation adds states where increased precision is required. OMNI by contrast starts with a structural model of interconnected activities whose temporal characteristics are described by observation data. The refinement process then replaces states in the model with more complex phase-type distribution models. In addition OMNI does not rely on simulation techniques to explore the high-level CTMC but instead uses probabilistic model checking.

Chapter 5

Evaluation of the OMNI Approach

5.1 OMNI refinement tool

We implemented OMNI as a Java tool that takes as input a high-level CTMC model. This model is specified in a variant of the PRISM modelling language [8] where state transition commands are expressed using components labels. For example, the PRISM command

$$\mathsf{s}=1 \;\rightarrow\; \mathsf{p}_1 * \lambda_1 : (\mathsf{s}'=2) + (1-\mathsf{p}_1) * \lambda_1 : (\mathsf{s}'=3);$$

that defines the outgoing transitions for state s_1 of our high-level CTMC model from Figure 3.2 is replaced by

$$\mathsf{s}=\langle \mathit{location} \rangle \;\rightarrow\; \mathsf{p}_1 : (\mathsf{s}'=\langle \mathit{arrival} \rangle) + (1-\mathsf{p}_1) : (\mathsf{s}'=\langle \mathit{departures} \rangle);$$

in the OMNI variant of the modelling language. This indicates that the CTMC transitions from the state associated with the *location* web service to either the state associated with the *arrival* web service (with probability p_1) or to the state associated with the *departures* web service (with probability $1-p_1$). An XML configuration file is then used to map each of these web service labels to a file of comma-separated values containing the observed execution times for the relevant web service. In addition, this configuration file allows the user to define the OMNI refinement parameters (i.e. k_i from Theorem 1, and α, $MinC$, $MaxC$, $MaxP$ and $MaxSteps$ from Algorithm 1). The *FittingAlg* parameter is fixed in the current version of the tool, so that OMNI uses the expectation-maximisation PHD fitting algorithm mentioned in Example 3.4.

When multiple QoS properties (for the same high-level CTMC) are provided to the OMNI tool, we avoid the overheads associated with the repeated execution of the modelling tasks from Table 4.2 for the same process activities by maintaining a cache of all completed tasks and their results. As such, each of these tasks is executed at most once per process activity, and its cached result is used when needed instead of repeating the task. By comparison, refining the whole CTMC indiscriminately for even a single QoS property would require the execution of these modelling tasks for every process activity.

Finally, to support the scenario where the delays (3.5) are negligible compared to the holding times (3.6), the configuration file allows the specification of a *delay threshold*, and activities with delays below this threshold are not included in the joint delay modelling step of the OMNI refinement. We found experimentally that this leads to significant reductions in the size of the refined CTMC with no impact on the accuracy of the QoS analysis.

Our OMNI tool uses the HyperStar PHD fitting tool from [83] for the CBFitting function from Algorithm 2, and produces the refined CTMCs as standard PRISM models. The OMNI tool is freely available from our project webpage `https://www.cs.york.ac.uk/tasp/OMNI/`, together with detailed instructions and all the models and datasets from this evaluation.

5.2 Evaluation

We evaluated OMNI by performing a set of experiments aimed at answering the following research questions.

RQ1 (Accuracy/No overfitting): How effective are OMNI models at predicting QoS property values for other system runs than the one used to collect the execution-time observation datasets for the refinement?

RQ2 (Refinement granularity): What is the effect of varying the OMNI refinement granularity on the refined model accuracy, size and verification time?

RQ3 (Training dataset size): What is the effect of the training dataset size on the refined model accuracy?

RQ4 (Component classification): What is the benefit of using an activity classification step within OMNI?

To assess the generality of OMNI, we carried out our experiments within two case studies that used real systems and datasets from different application domains. The first case study is based on the travel web application presented as a motivating example in Section 3.1. In the second case study, we applied OMNI to an IT support system. This system is introduced in Section 5.2.1, followed by descriptions of the experiments carried out to address the four research questions in Sections 5.2.2–5.2.5.

5.2.1 IT support system

The real-world IT support system we used to evaluate OMNI is deployed at the Federal Institute of Education, Science and Technology of Rio Grande de Norte (IFRN), Brazil. The system enables the IFRN IT support team to handle user tickets reporting problems with the institute's computing systems. As part of our collaboration with IFRN researchers [2], system logs covering the handling of 1410 user tickets were collected from this IT support system over a period of six months between September 2016 and February 2017.

A high-level abstract CTMC model of the business process implemented by the IT system

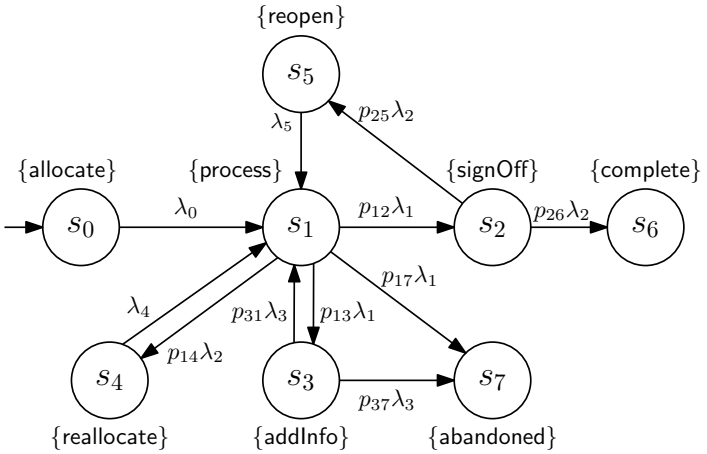


Figure 5.1: High-level CTMC model of IT support system

is shown in Figure 5.1. In this model, state s_0 corresponds to a ticket being created by a "client" and awaiting allocation to a member of the support team. Once allocated, the ticket is processed (state s_1) and, if the issue can be resolved, the client is informed and the ticket awaits sign off (s_2) before being marked as complete (s_6). The client may choose to reopen (s_5) the ticket rather than close it, in which case the ticket is returned to the support team member for further processing. Whilst processing a ticket, the support staff may require additional information from the client (s_3) or may need to reallocate the ticket to another member of the IT support team (s_4). A ticket may also be abandoned (s_7) either during processing or whilst awaiting additional information from the client.

We used our OMNI tool to refine the high-level CTMC from Figure 5.1 in order to support the verification of the response time of the IT support system through the analysis of two properties:

$$\begin{array}{ll} \textbf{P1} & P_{=?}[F^{[0,T]}\, complete] \\ \textbf{P2} & P_{=?}[(\neg reopen\ \&\ \neg addInfo)\ U^{[0,T]}\, complete] \end{array} \tag{5.1}$$

where **P1** specifies the probability of a ticket reaching the complete state within T (working) hours, and **P2** represents the probability of ticket handling being completed within T working hours without further input from the client who raised the ticket and without the ticket being reopened.

We used only half of the six-month logs (covering 705 tickets created over the first approximately three months) for the OMNI refinement, so that we could use the other half of the logs to answer research question RQ1 (cf. Section 5.2.2.2). For each ticket, the time spent in a particular state was derived from the log entries, taking into account only the working hours for the IT support team.[1] Assuming exponentially distributed execution times for the components of the IT support process, we used (3.1) to calculate the component execution rates shown in Table 5.1. Finally, we used the logs to calculate the frequencies of state transitions, and thus to estimate the CTMC state transition probabilities as shown in Table 5.2.

[1]The working hours for the period covered by the logs were identified through consultation with the IFRN owner of the IT support process.

Table 5.1: Execution rates for the IT support system

Component	**Rate** (hours^{-1})
allocate	$\lambda_0 = 0.08248$
process	$\lambda_1 = 0.09799$
signOff	$\lambda_2 = 0.01167$
addInfo	$\lambda_3 = 0.02006$
reallocate	$\lambda_4 = 0.02839$
reopen	$\lambda_5 = 0.09988$

Table 5.2: Transition probabilities for the IT support system

CTMC states		**Transitions**	**Transitions**	**Estimate transition**
s_i	s_j	**from** s_i **to** s_j	**leaving** s_i	**probability** $s_i \to s_j$
s_1	s_2	533	705	$p_{12} = {}^{533}/_{705} = 0.76$
s_1	s_3	24	705	$p_{13} = {}^{24}/_{705} = 0.03$
s_1	s_4	34	705	$p_{14} = {}^{34}/_{705} = 0.05$
s_1	s_6	114	705	$p_{16} = {}^{114}/_{705} = 0.16$
s_2	s_6	501	533	$p_{25} = {}^{501}/_{533} = 0.94$
s_2	s_5	32	533	$p_{26} = {}^{32}/_{533} = 0.06$
s_3	s_1	16	24	$p_{31} = {}^{16}/_{24} = 0.67$
s_3	s_7	8	24	$p_{37} = {}^{8}/_{24} = 0.33$

Figure 5.2 compares the actual values of properties **P1** and **P2** from (5.1) – computed based on the system logs – with the values predicted by the analyses of: (a) the high-level CTMC from Figure 5.1 (with the parameters given in Tables 5.1 and 5.2); and (b) OMNI-refined CTMC models for the two properties. The refined CTMCs were obtained using the same OMNI parameters as in Example 3.4, except $\alpha = 0.2$ and a *delay threshold* of 0.01 hours.[2] As explained in Section 5.1, this threshold meant that components with a delay (3.5) below 0.01 hours (which amounted to all component of the IT support system) were not included in the joint delay modelling of OMNI.

Having introduced the system used in our second case study, we will use the next sections to describe the experiments carried out to answer our four research questions.

5.2.2 RQ1 (Accuracy/No overfitting)

The generation of OMNI-refined CTMC models requires the processing of finite datasets produced by the components of the analysed system, in order to extract key model features. To be useful, these CTMCs should accurately predict the values of the system properties for other system runs, i.e. should not be overfitted to the datasets used to generate them.

[2]The threshold value was chosen to be approximately three orders of magnitude smaller than the smallest mean execution time of a system component, i.e. $1/\lambda_5 = 10.012$ hours for the IT support system (cf. Table 5.1).

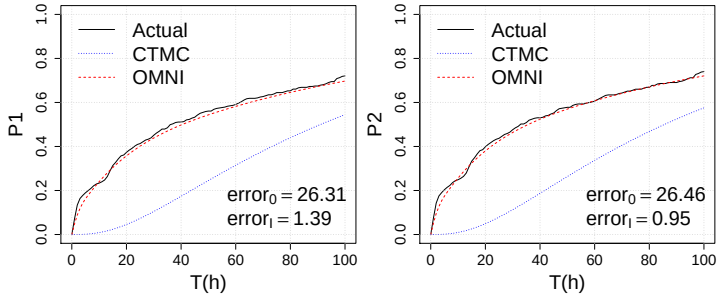


Figure 5.2: Actual values of the IT support system properties versus property values predicted using the high-level and the refined CTMC models, over 100 working hours from ticket creation; the prediction error (3.4) for the refined CTMCs (i.e. $\mathsf{error}_\mathsf{I}$) is 94.7% smaller (for property **P1**) and 97% smaller (for property **P2**) than the corresponding prediction errors for the high-level CTMC (i.e. error_0).

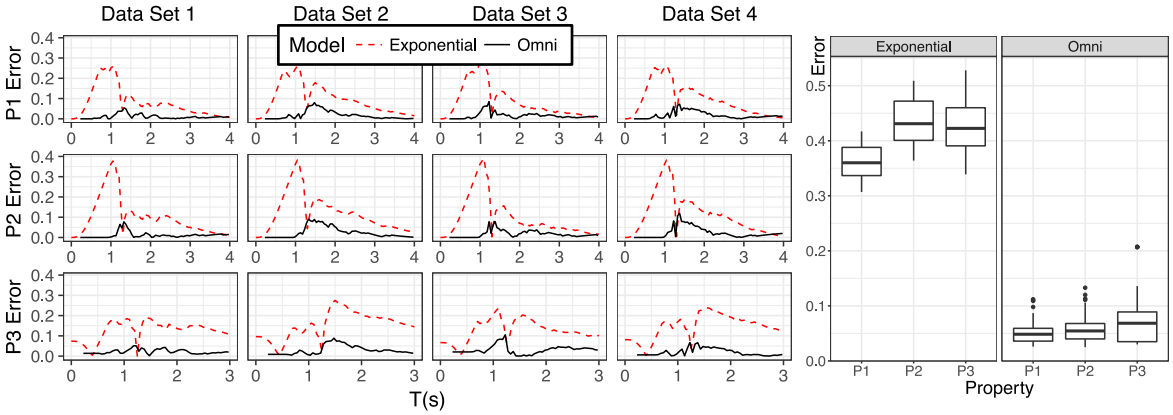


Figure 5.3: Prediction error for the web application properties, for training and testing datasets from different runs

5.2.2.1 Travel web application

To assess whether the OMNI web application models accurately predict the value of system properties without overfitting, we obtained three additional datasets (labelled 'Data Set 2', 'Data Set 3' and 'Data Set 4') for the travel web application. Each new dataset corresponds to a four-hour run, with all datasets (including the original dataset, 'Data Set 1') captured over a period of two days. Figure 5.3 (a) shows the difference between the property values predicted by the CTMC analysis and the actual property values taken from each of the four datasets. Results are shown for the initial CTMC from Figure 3.2 (labelled 'Exponential' in the diagrams) and the refined models obtained using 'Data Set 1' (labelled 'Omni' in the diagrams). In all cases, the OMNI-refined CTMCs significantly improve the accuracy of the analysis when compared to the traditional CTMC analysis approach.

To ensure that the reduction in error was not due to a favourable partition of the dataset, a further set of experiments was carried out. All four available datasets were combined into a single dataset from which four new disjoint subsets were created by random sampling

without replacement. One of the resulting datasets was designated the training set and used to create a high level CTMC and OMNI-refined CTMCs for each property. These models were then tested against the remaining datasets to calculate the cumulative error (3.4) for each property. The process was repeated 30 times, and the results - presented as box plots in Figure 5.3(b) - show that the refined CTMCs always improved the accuracy of the predictions by a wide margin.

5.2.2.2 IT support system

For the IT support system, OMNI-refined CTMCs for the two properties were produced from half of the available system logs ('Data Set 1') as described in Section 5.2.1. We then assessed the accuracy of the predictions obtained using these refined CTMCs against the actual property values extracted from the training set ('Data Set 1') and from the test dataset ('Data Set 2') produced from the second half of the system logs. Figure 5.4(a) shows the error when the predicted values are compared to actual values for the two datasets. For both datasets, the OMNI-refined CTMCs produce results which significantly outperform the results obtained by analysing the high-level CTMC.

As for the first case study, we performed additional experiments to confirm that the positive results from Figure 5.4(a) were not due to a favourable partition of the available observations. To this end, we combined the two datasets and randomly partitioned them into new training and test datasets of equal size. A high-level CTMC and OMNI refined CTMCs were then created using the new training dataset, and all models were used to predict the values of the two properties for the test dataset. This experiment was repeated 30 times and each time the cumulative errors in the predictions were calculated for each property. The box plots in Figure 5.4(b) summarise these experimental results, confirming the improved accuracy of the OMNI-refined CTMCs.

5.2.2.3 Discussion

The experiments described in the previous sections show that OMNI consistently outperformed the traditional CTMC modelling and analysis approach in both case studies, irrespective of the choice of training set. This confirms that OMNI models can effectively predict QoS property values for other system runs than the one used to collect the training datasets employed in the refinement.

Our experiments also showed that the error profiles from Figures 5.3 and 5.4 capture several general features for the type of QoS analysis improved by OMNI:

1. The initial peak in the 'Exponential' prediction error for properties **P1** and **P2** from Figure 5.3 is characteristic of the inability of exponential distributions to model delays, as also explained in Section 3.2.2. OMNI does not suffer from this limitation. Note that this modelling error does not affect properties **P1** and **P2** from Figure 5.4 because the delays for the IT support system are insignificant compared to the holding times.
2. The second peak in the 'Exponential' prediction error for properties **P1** and **P2** from Figure 5.3, and the first peak for properties **P1** and **P2** from Figure 5.4 are representa-

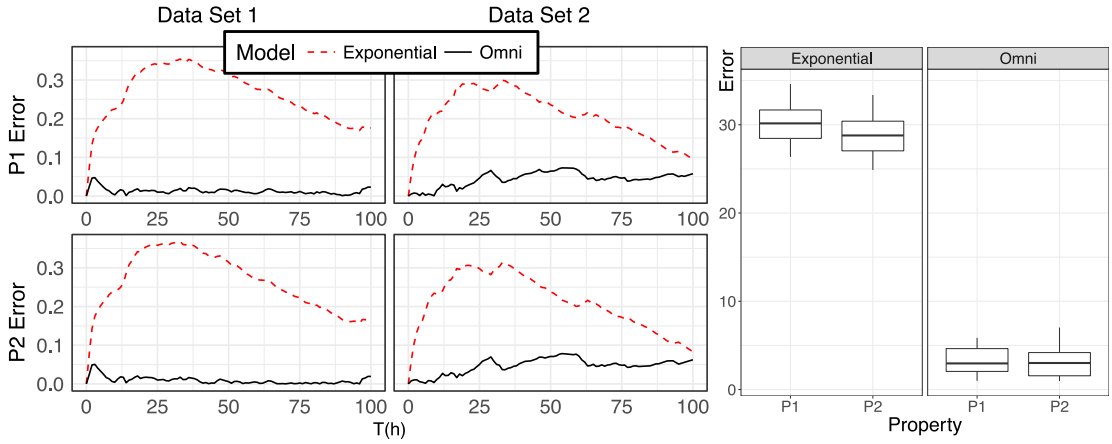


Figure 5.4: Prediction error for the IT support system properties, for training and testing datasets from different three-month time periods

tive of the inability of exponential distributions to model long tails (due to operations occasionally having much longer execution times than their typical execution times). As such, the estimated rates of the exponential distributions are too low, and the predictions are overly conservative. These error peaks are particularly high (above 0.3) for the IT support system, as IT support personnel occasionally required very long times to address a user request. Again, OMNI yields much smaller prediction errors around these peaks.

3. The multiple peaks in the 'Exponential' prediction error for property **P3** from Figure 5.3 is characteristic of derived properties, i.e., properties defined using multiple "primitive" properties (in this case, **P3** represents profit, and is defined as the difference between revenue and penalties). The multiple peaks are due to the prediction errors for the primitive properties peaking at different time moments. As before, OMNI significantly dampens these peaks.

5.2.3 RQ2 (Refinement granularity)

To evaluate the effects of refinement granularity we constructed a set of OMNI models by varying:

1) k_i, the number of states in the Erlang delay models from the joint delay modelling of OMNI (cf. Theorem 5);
2) α, the PHD model fitting threshold used in the holding time modelling of OMNI (cf. Algorithm 1).

Larger values of k_i are associated with increased accuracy in the modelling of delays, whilst reducing α corresponds to finer-grained refinement in the PHD modelling.

Table 5.3: Effects of the OMNI refinement granularity on web application model

		P1			P2			P3		
k_i	α	**#states**	**Error**	$T_V(s)$	**#states**	**Error**	$T_V(s)$	**#states**	**Error**	$T_V(s)$
Initial CTMC		7	0.325	1.9	7	0.402	1.9	7	0.377	1.9
10	0.2	82	0.085	3.9	45	0.126	3.5	82	0.094	5.0
100	0.2	262	0.049	5.6	135	0.066	4.3	262	0.077	7.7
259	0.2	580	0.045	8.8	294	0.060	5.8	580	0.074	12.6
10	0.1	232	0.078	6	118	0.112	4.3	232	0.083	8.1
100	0.1	412	0.043	7.8	208	0.049	5.1	412	0.063	11.0
259	0.1	730	0.038	11.4	367	0.042	6.8	730	0.059	16.5
10	0.05	618	0.075	13.8	376	0.106	7.9	618	0.081	19.6
100	0.05	798	0.041	16.0	466	0.044	8.8	798	0.061	22.7
259	0.05	1116	0.036	20.6	625	0.036	10.8	1116	0.057	29.4

5.2.3.1 Travel web application

The experimental results from the web application case study are presented in Table 5.3. As k_i is increased from 10 to 100 and from 100 to 259, the error is reduced.[3] However, this improvement shows diminishing returns for all properties as k_i becomes large. The same pattern occurs as α is decreased, with smaller errors for smaller α values but only a marginal reduction in error as α is reduced from 0.1 to 0.05.

Since k_i controls the number of states associated with delays, increasing k_i also increases the total number of states associated with the model. The model size also increases as α is decreased.

Finally, the experimental results confirm that the models for property **P2** are consistently much smaller than for **P1** and **P3** since more states from the initial CTMC are in the "exclude from refinement" set S_{X} when evaluating **P2** than when evaluating the other properties (cf. Table 4.1). T_V is the total time for PRISM to verify each property in the interval $[0, T_{\max}]$ with a time step of 0.05s and includes the time taken for model construction. All experiments presented here were carried out on a MacBook Pro with 2.9 GHz Intel i5 processor and 16Gb of memory. As the model increases in size, and accuracy improves, the time taken for verification also increases, up to 29.4s for the finest-grained model used to evaluate property **P3** across the entire interval $[0, T_{\max}]$.

5.2.3.2 IT support system

When OMNI is applied to the IT support system, the delay threshold of 0.01 hours chosen as explained in Section 5.2.1 means that the delay modelling was omitted (i.e. delays were approximated to zero). As such, we were not interested in varying k_i in this case study, and Table 5.4 only shows the effects of decreasing α on the refined models. Like in the first case study, decreasing α gradually reduces the prediction error, with a significant error reduction obtained even for the largest α from our experiments (e.g. an over tenfold reduction from 26.3

[3] These k_i values are taken from Table 3.2 on page 48.

Table 5.4: Effects of the OMNI refinement granularity on the IT support system model

	P1			P2		
α	**#states**	**Error**	$T_V(s)$	**#states**	**Error**	$T_V(s)$
Initial CTMC	8	26.3	3.0	8	26.5	3.0
0.6	44	2.45	42.8	41	2.11	41.7
0.4	61	2.24	48.2	51	1.74	44.4
0.2	202	1.39	95.8	174	0.95	84.4
0.1	329	1.27	139.6	259	0.87	110.6
0.05	722	1.01	274.6	346	0.84	139.8

for the initial, high-level CTMC and property **P1** to just 2.45 for the coarsest-granularity CTMC generated for $\alpha = 0.6$). Diminishing returns in terms of error reduction are achieved for property **P2**; for **P1**, this trend is not clearly distinguishable for the tested α values.

As expected, the model size grows as α is decreased, leading to a corresponding increase in the verification time T_V. T_V includes the time for the construction of the model and for PRISM to analyse the property in the interval $[0, 100\text{h}]$ with a time step of one hour (i.e. 100 verification sessions). The largest verification time is 274.6s for the finest-granularity CTMC obtained for property **P1**, which is entirely acceptable for an offline verification task.

During the activity classification step of OMNI, the exclusion sets for the two properties are calculated as $S_{\mathsf{X}} = \{s_6, s_7\}$ for **P1** and $S_{\mathsf{X}} = \{s_3, s_5, s_6, s_7\}$ for **P2**. Therefore, the models associated with **P2** are consistently smaller than those associated with **P1**, whose exclusion set S_{X} contains only two states.

5.2.3.3 Discussion

For both case studies and all considered QoS properties, considerable improvements in model accuracy are obtained even with small, coarse-grained OMNI models. As such, OMNI can offer significant improvements in accuracy over traditional CTMC modelling techniques even when computational resources are at a premium. Additional, but typically diminishing, gains in prediction accuracy are obtained through increasing the granularity of the refinement. Expectedly, this leads to a corresponding increase in verification time. For our two systems, this time did not exceed 10 minutes (and was typically much smaller) for all considered properties and model granularities – an acceptable overhead for the offline verification task performed by OMNI. The selection of values for k can be made with respect to the maximum error one is willing to accept in delay modelling, as shown in Section 3.2.2. Selecting a value for α, which produces an acceptable trade off between model accuracy and time to compute, may be more difficult and our case studies have shown that acceptable values vary by application domain. The development of a heuristic for the selection of α, from the observed data and high level CTMC structure, presents an area for further work.

Table 5.5: Web application – 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%††	0.038 sd N/A*	0.042 sd N/A*	0.059 sd N/A*
80%	0.038 sd 0.006	0.043 sd 0.005	0.058 sd 0.023
60%	0.046 sd 0.013	0.048 sd 0.014	0.078 sd 0.041
40%	0.057 sd 0.017	0.063 sd 0.022	0.105 sd 0.054
20%	0.083 sd 0.032	0.076 sd 0.026	0.156 sd 0.075
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

5.2.4 RQ3 (Training dataset size)

In both case studies, we ran a set of experiments to evaluate the effect of reducing the training dataset size on the accuracy of OMNI models. For each experiment, training subsets were constructed by randomly selecting a percentage of all available datasets used to answer the previous research questions. The sizes of these selected subsets were 80%, 60%, 40% and 20% of the complete training dataset from Sections 5.2.2.1 and 5.2.2.2. For each system and each of its analysed QoS properties, the experiments were repeated 30 times, with the property errors recorded.

5.2.4.1 Travel web application

Table 5.5 shows the mean error and standard deviation (labelled 'sd') for the web application case study with $k_i = 259$ and $\alpha = 0.1$. As the training dataset size decreases, the error and standard deviation associated with each property show an increasing trend. However, we note that at 80% the prediction errors show little difference to the 100% figures – the mean errors at 100% are very close to the 80% errors, and well within one standard deviation of the 80% mean. This suggests that 80% of the complete dataset is sufficient to capture the characteristics of the underlying component distributions for this case study.

5.2.4.2 IT support system

For the IT support system, the experimental results are provided in Table 5.6. As for the other case study, we observe that reducing the size of the training sets leads to a trend where the prediction error and the standard deviation increases. This also happens when the size of the training dataset is reduced from 100% to 80%, suggesting that additional slight improvements may be possible by further increasing the size of the initial training dataset.

Table 5.6: IT support system – training dataset size effect on prediction accuracy, shown as average error and standard deviation over 30 runs

Dataset†	P1 Error	P2 Error
100%††	1.39 sd N/A*	0.95 sd N/A*
80%	1.53 sd 0.774	1.35 sd 0.570
60%	1.57 sd 0.942	1.55 sd 0.797
40%	2.37 sd 1.450	2.19 sd 1.396
20%	3.70 sd 2.825	3.81 sd 3.039
Initial CTMC	26.31 sd N/A*	26.46 sd N/A*

†Percentage of complete 705-element training dataset
††Single run using entire data set
*Single run, so no standard deviation

5.2.4.3 Discussion

For both case studies we note that even modest training dataset sizes show a significant improvement over the traditional approach to CTMC-based analysis of QoS properties. For the web application, a training set consisting of 20% of the original dataset equates to only 54 request handling observations, and reduces the mean estimation error by between 50–81% for the properties of interest. For the IT system, 20% of the original training dataset equates to 141 tickets processed, with the processing of only five tickets using the addInfo component of the system, yet the prediction errors for **P1** and **P2** are both reduced by approximately 86%.

5.2.5 RQ4 (Property-centric refinement)

We evaluated the effects of extending the CTMC-refinement method from Chapter 3 with the property-centric refinement step described in Chapter 4. To this end, we performed experiments to compare the model size, verification time and accuracy of the refined CTMCs generated by the complete OMNI method described and refined CTMCs which did not use property-centric refinement. A version of the OMNI tool which excluded property-centric refinement was developed and presented in [3] and is used here for comparison with the newer, complete OMNI solution.

5.2.5.1 Travel web application

For the web application, refined CTMCs were built using first OMNI method from Chapter 3 and then the complete OMNI method which considers the property to be refined, initially with parameters $k_i = 259$ and $\alpha = 0.1$. The first two rows from Table 5.7 summarise these experimental results, which show that the use of property-centric refinement yields significant reductions in the number of model states and the verification time for all three properties compared to the method without this step. As expected given the CTMC state partition from Table 4.1 and OMNI rules from Table 4.2, the largest reductions are achieved

for property **P2** (79% fewer model states, and 71% shorter verification time). For properties **P1** and **P3**, the use of property-centric refinement led to a reduction in model size of over 58%, and to reductions in verification time of 54% and 57%, respectively.

The prediction errors are very close for both OMNI variants, and considerably smaller than the errors for the high-level CTMC (provided in the last row of Table 5.7 for convenience). However, the errors are negligibly larger when property-centric refinement is used. This is due to the use of fewer states for OMNI's joint delay modelling compared to the separate modelling of component delays in [3]. As shown in Table 5.3, additional reductions in prediction error may be achieved by increasing k_i or reducing α if required. For example, by setting $\alpha = 0.05$, the refined models still have much fewer states than the preliminary OMNI model, show an improvement in verification time of between 17–54%, and are more accurate. The third row of Table 5.7 shows again these experimental results for ease of comparison.

5.2.5.2 IT support system

For the second case study, the experimental results are presented in Table 5.8. Whilst our fully fledged OMNI allows for component delays to be omitted from the refinement when they are below a delay threshold (cf. Section 5.1), this was not possible in the initial OMNI tool presented in [3]. Therefore, to ensure a fair comparison, we used a small k_i value ($k_i = 10$) when generating refined CTMCs with the preliminary OMNI variant. As shown by the experimental results, using the fully fledged OMNI yields smaller refined models that take significantly less time to verify for both of the IT system properties. Furthermore, these smaller refined models achieve the same prediction accuracy as the larger models generated by the preliminary OMNI.

5.2.5.3 Discussion

The property-centric refinement step which the OMNI method employs in its model construction uses the high-level CTMC model only and the time taken for its execution is very small. For the web application, the time taken to classify the high-level CTMC states for all three properties was 2.3s, and for the IT support system this step took only 1.8s.

For both case studies presented we have shown that for all the properties considered it was possible to generate OMNI models which are smaller, faster to verify and no less accurate than those produced when property-centric refinement is omitted. The amount of verification time saved depends on the number of states for which delays can be combined, and on the number of states which can be excluded from refinement – but these savings were considerable in all our experiments with the two real-world systems.

Table 5.7: Web application – comparison of OMNI with and without property centric refinement

	P1			P2			P3		
Model	**#**	**Error**	$T_V(s)$	**#**	**Error**	$T_V(s)$	**#**	**Error**	$T_V(s)$
OMNI no classification ($\alpha = 0.1$, $k_i = 259$)	1766	0.037	24.95	1766	0.039	23.79	1766	0.063	38.21
OMNI ($\alpha = 0.1$, $k_i = 259$)	730	0.038	11.40	367	0.042	6.80	730	0.059	16.5
OMNI ($\alpha = 0.05$, $k_i = 259$)	1116	0.036	20.6	625	0.036	10.8	1116	0.057	29.4
High-level CTMC	7	0.325	2.53	7	0.402	2.50	7	0.377	2.47

Table 5.8: IT support system – comparison of OMNI with the preliminary CTMC refinement approach from [3]

	P1			P2		
Model	**#states**	**Error**	$T_V(s)$	**#states**	**Error**	$T_V(s)$
Preliminary OMNI ($\alpha = 0.2$, $k_i = 10$)	265	1.39	261.5	265	0.95	239.6
OMNI ($\alpha = 0.2$)	202	1.39	95.8	174	0.95	84.4
High level CTMC	8	26.3	3.0	8	26.5	3.0

5.3 Threats to validity

5.3.1 External validity

External validity threats may arise if the stochastic characteristics of the systems from our case studies are not indicative of the characteristics of other systems. To mitigate this threat, we used two significantly different systems from different domains for the OMNI evaluation. The section labelled 'System' from Table 5.9 summarises the multiple characteristics that differ between these systems.

In addition, the datasets used in the two case studies present different characteristics, as shown in the 'Datasets' section from Table 5.9. In particular, the datasets for the service-based system were obtained from real web services, while for the IT support system they were taken from the actual system logs. This gives us confidence that the stochastic characteristics of the two systems (including regions of zero density, multi-modal response times, and long tails) are representative for many real-world systems.

The next section from Table 5.9 summarises the different types of QoS properties analysed in our case studies. The transient fragment of continuous stochastic logic (whose analysis accuracy is improved by OMNI supports, cf. Section 2.2.2.3) supports the specification of multiple classes of QoS properties of interest, including success probability, profit/cost and response time, and our case studies considered examples of all of these.

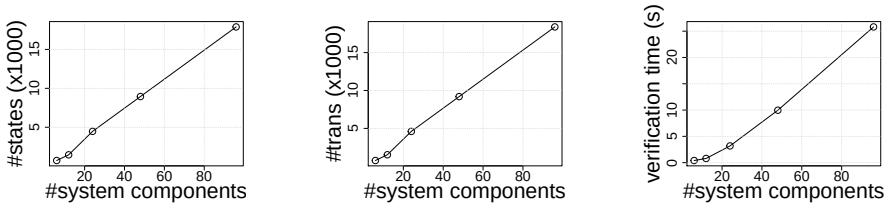


Figure 5.5: Refined CTMC states, transitions and verification time for property **P1** at a single time point, for system sizes up to 16 times larger than the web application

Finally, as shown in the 'Purpose of QoS analysis' section from Table 5.9, the first case study applied OMNI during the design stage of the development process, whereas the second case study assessed OMNI by verifying QoS properties of an existing system.

Another external threat may arise if the OMNI-refined CTMC models were too large to be verified within a reasonable amount of time. The OMNI approach mitigates this threat by allowing for the refinement to be carried out at different levels of granularity, and our experiments indicate that significant improvements in prediction accuracy is achievable with modest enlargement of the models.

Our two case studies are based on real systems, but these systems have a relatively small number of *modelled activities*. For the large IT system from the second case study, the small number of *modelled activities* is due to model abstraction such that the model only includes activities that influence the analysed QoS properties.

For systems with larger numbers of modelled activities, we note that the increase in model size due to the OMNI refinement is only linear in the number of activities to be modelled. Moreover, as OMNI uses acyclic PHDs, the number of transitions also increases linearly. Modern model checkers can handle CTMCs with a large number of states and as such we expect OMNI to scale well with much larger systems. We confirmed these hypotheses by constructing models with 12, 24, 48 and 96 activities by combining 2, 4, 8 and 16 instances of our web application CTMC from Figure 3.2.[4] OMNI was then used to refine the composite models with $k_i = 259$ and $\alpha = 0.1$. For each refined CTMC, we measured the number of states, the number of transitions, and the time taken to verify property **P1** of the travel web application at the single time point $T = 20s$ (since **P2** and **P3** can not be meaningfully extrapolated to these larger systems). The results of these experiments, shown in Figure 5.5, confirm the predicted linear increase in the verification overhead with the system size.

5.3.2 Construct validity

Construct validity threats may be due to the assumptions made when collecting the datasets or when defining the QoS properties for our model refinement experiments. To address the first threat, we collected the datasets from a real IT support system and from a prototype web application that we implemented using standard Java technologies and six real web

[4]We did not perform similar experiments for the IT support system as they would not have been qualitatively different.

services from three different providers. For the first system, the datasets were collected over a period of six months, and for the second system they were collected on two different days, at different times of day. Furthermore, we used different datasets for training and testing. To mitigate the second threat, we analysed three performance and cost properties of web application, and two typical performance properties of the IT system.

5.3.3 Internal validity

Internal validity threats can originate from the stochastic nature of the two analysed systems or from bias in our interpretation of the experimental results. We addressed these threats by providing formal proofs for our CTMC refinement method, by reporting results from multiple independent experiments performed for different values of the OMNI parameters, and by analysing several QoS properties at multiple levels of refinement granularity. Additionally, we made the experimental data and results publicly available on our project webpage in order to enable the replication of our results.

Table 5.9: Characteristics of the case studies used to evaluate OMNI

Characteristic	**Case study 1**	**Case study 2**
	System	
Type of system	Prototype service-based system developed by the OMNI team	Production IT support system developed by, running at, and managed by university in Brazil
System components	Mix of six high-performance (commercial) and budget (free) third-party web services invoked remotely over the Internet	Proprietary software components deployed on university computing infrastructure, and supporting human tasks with high variance in temporal characteristics
Size of system	3188 lines of code	1276131 lines of code
Component execution times	Tens to hundreds of milliseconds	Minutes to hours
Operational profile	Assumed values for the probabilities of the different types of requests	Probabilities of different operation outcomes extracted from the real system logs
	Datasets	
Dataset source	Obtained from invocations of real web services	Taken from actual system logs
Key dataset features	Significant delays (compared to holding times) due to network latency	Long tails and outliers due to a small number of complex user tickets; multi-modal response times and regions of zero density due to different experience levels of IT support personnel; negligible delays
	Analysed QoS properties	
Types of properties	Overall success probability (**P1**) Success probability for "day-trip" requests (**P2**) Profit = revenue − penalties (**P3**)	Overall response time (**P1**) Response time for "straightforward" user tickets (**P2**)
	Purpose of QoS analysis	
Supported stage of development process	Design of new system	Verification of existing system

Part III

Discrete Time

Chapter 6

Observation-Enhanced DTMC Verification

Quantitative verification of discrete time systems allows for mathematically provable guarantees of operational correctness. The development of tools such as PRISM [8] and MRMC [9] has led to the adoption of these techniques in a range of applications [103]. The usefulness of such techniques is, however, limited by the accuracy of the discrete time Markov models on which the analysis depends. Whilst states and transitions within these model may easily be identified (e.g. from process diagrams or systems analysis techniques), obtaining accurate values for model parameters is often difficult and, where parameter inaccuracies exist, the analysis of such models may be misleading.

When considering model parameters we may classify the uncertainty associated with them into two major categories [104]: **Epistemic** uncertainty arises from a lack of knowledge of system components or the environment in which the system operates. This type of uncertainty may be reduced as our knowledge of the system under consideration increases. Eliminating such uncertainty may, however, be impracticable. **Aleatory** uncertainty is associated with the inherent stochastic nature of a system, process, or environment, under consideration. This category of uncertainty cannot be reduced.

In this chapter we consider how observation data may be used to parametrise a discrete time Markov model and how traditional assumptions for such parameter estimates may produce misleading results. We then present techniques which allow for more accurate analysis of the Markov models. Using these techniques we develop a probabilistic model checker for formal verification with confidence intervals, FACT. In the next chapter we show how the analysis of models using FACT can be used to identify abnormal behaviour from logged data traces of system processes.

The common practice for obtaining parameter estimates is through domain expertise, or by model fitting to log data or run-time observations [105],[71]. Ghezzi et al [70] demonstrated how this approach may be used to extract models of user activity from log data. The technique which they call BEAR also allows for states to be added to the model as they are identified in the log.

Estimating parameters using a maximum likelihood estimators (MLE) is a well established technique [106]. Where the transition probabilities are stationary the maximum likelihood estimate for the probability of transitioning from state i to state j is given as

$$p_{ij} = \frac{n_{ij}}{n_i} \tag{6.1}$$

where n_{ij} is the number of observed transitions from state i to state j and n_i is the total number of transitions observed to leave state i.

These estimates, however, contain estimation errors which are then propagated and amplified by quantitative verification since Markov models are non-linear. This then leads to imprecise results that can in turn yield invalid design decisions or verification conclusions.

One problem with using the estimator presented in (6.1) is that it does not tell us how much information the estimate is based on or the potential size of the error. Intuitively we know that using two samples to create our estimate will be less accurate than using 1000 samples and hence a point estimate is insufficient to capture our confidence in the estimation. Indeed generating point estimates from successive sample sets will yield a set of non identical estimates meaning that we are unsure which, if any, is correct.

One alternative is to calculate an interval estimator which provides a bound within which we expect the estimate to lie. The use of confidence intervals for parameters which are characterised as stochastic processes was first proposed by Neyman [107] and have become widely used as a method for inferring parameter values from sample data [108],[109],[110].

An interval estimate of the value θ is then of the form

$$\hat{\theta}_1 < \theta < \hat{\theta}_2 \tag{6.2}$$

where $\hat{\theta}_1$ and $\hat{\theta}_2$ are upper and lower bounds such that for some specified probability $(1-\alpha)$, θ exists in the range specified by the interval.

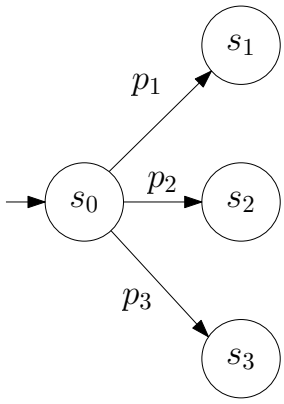
$$P(\hat{\theta}_1 < \theta < \hat{\theta}_1) = (1-\alpha) \tag{6.3}$$

where $\hat{\theta}_1$ and $\hat{\theta}_2$ define our confidence interval and $(1-\alpha)$ is our confidence level [108].

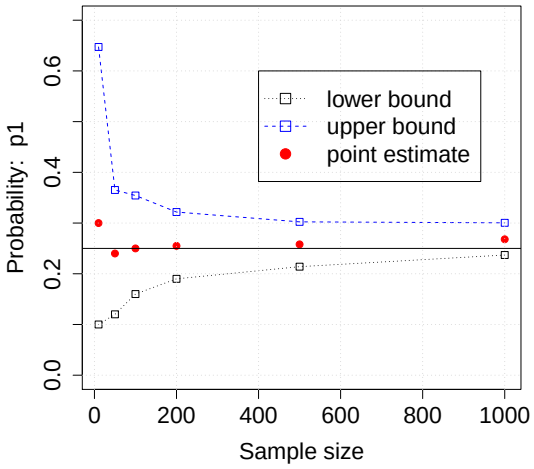
Example 6.1. *Let us consider the state transition model shown in Figure 6.1a. Starting in state s_0 the system will transition to one of three states s_1, s_2 or s_3 with probability $p_1 = 0.25, p_2 = 0.6$ and $p_3 = (1 - p_2 - p_3) = 0.15$ respectively. If we assume that the absolute values of the probabilities are unknown but a count of transitions is available then a point estimate for each probability may be calculated using (6.1).*

Six sets of observations were obtained with sample sizes $\{10, 50, 100, 200, 500, 1000\}$. For each sample set a point estimate for each probability (p_1, p_2, p_3) was calculated and the results plotted in Figure 6.1(b) – (d). As the number of samples increases the point estimate generally gets closer to the true value, however we can not guarantee this and indeed examining the absolute errors in Table 6.1 shows that the point estimate for 1000 samples is less accurate than that obtained with 500 samples for both p_1 and p_2.

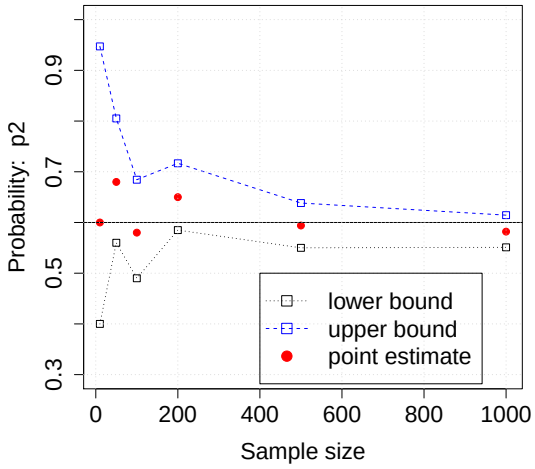
The state transition process is described by a multinomial distribution [108] and as such we



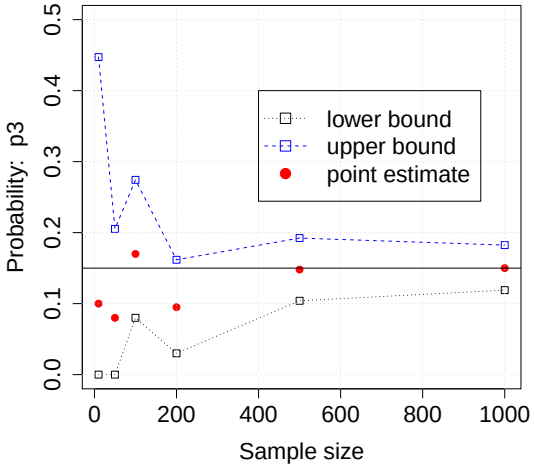
(a) State transitions from s_0



(b) Estimation of p_1



(c) Estimation of p_2



(d) Estimation of p_3

Figure 6.1: Estimating multinomial probabilities using sample sets of varying size.

calculate simultaneous confidence intervals [111] for the three probabilities which characterise the process[1]*. These intervals, calculated with a confidence level of* 0.95*, are then reported in Figure 6.1(b) – (d). The width of these intervals can be seen to reduce in size as the number of samples increases and, for this experiment, includes the true value for all probabilities in all sample sets.*

The remainder of this chapter is structured as follows. Section 6.1 summarises a method for the formal verification of discrete time Markov models with confidence intervals as developed by Calinescu et al. [23]. Section 6.2 then describes FACT, a probabilistic model checking tool which implements this method. The tool is then evaluated in Section 6.4. Finally Section 6.5

[1] In this case we use the MultinomialCI package in R.

Table 6.1: Absolute error associated with point estimate of stochastic process

sample size	$\mid \hat{p_1} - p_1 \mid$	$\mid \hat{p_2} - p_2 \mid$	$\mid \hat{p_3} - p_3 \mid$
10	0.050	0.000	0.050
50	0.010	0.080	0.070
100	0.000	0.020	0.020
200	0.005	0.050	0.055
500	0.008	0.006	0.002
1000	0.018	0.018	0.000

describes an overview of related work.

6.1 Background

Formal verification with confidence intervals [23] makes use of a parametric Markov chain (PMC) for which some of the transition probabilities are unknown and establishes confidence intervals for QoS properties described in PCTL. Given a PCTL property, Φ, parametric model checking [112],[113],[114],[115] produces a symbolic expression of Φ in which the PMC parameters appear.

The approach then takes as input:

1. a parametric Markov chain describing the system to be analysed $\mathcal{D} = (S, \boldsymbol{P}, \boldsymbol{\pi}_0, AP, L)$.
2. a PCTL formula Φ for a QoS requirement of interest where Φ may be a probabilistic state formula or a cost/reward formula.
3. A confidence level $(1 - \alpha)$ where the error level $\alpha \in (0, 1)$
4. a set O of observations of outgoing transitions from the states of the model $\mathcal{D}$ which are associated with unknown transition probabilities. For each state $s \in S$, O contains the tuple
$$obs_i = (n_{i1}, n_{i2}, \cdots, n_{in}) \tag{6.4}$$
where n_{ij} is the number of observed transitions from state s_i to state s_j.

To establish if a PCTL state formula $\Phi = \mathcal{P}_{\bowtie p}[\Psi]$ is satisfied we calculate the actual probability that Ψ is satisfied and then compare it with the bound p. We therefore calculate a confidence interval $[a, b]$ for $\mathcal{P}_{=?}[\Psi]$ with a confidence level of $(1 - \alpha)$ such that the interval calculated guarantees that the probability of any path which satisfies Ψ being outside this interval is less that α.

$$\text{Prob}\{Pr_{s_0}\{\pi \in Paths^{\mathcal{D}}(s_0) | \pi \vDash \Psi\} \notin [a, b]\} < \alpha \tag{6.5}$$

In order to evaluate $[a, b] \bowtie p$ relational operator rules are established between an interval

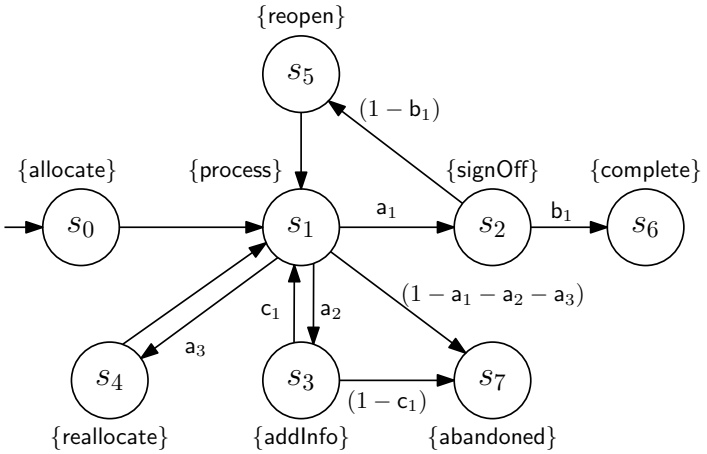


Figure 6.2: A parametric Markov chain model of an IT support system.

$[a, b] \subset \mathbb{R}$ and a scalar $x \in \mathbb{R}$ where $\bowtie \in \{\leq, <, >, \geq\}$ and:

$$\begin{aligned} [a, b] < x \text{ iff } b < x \\ [a, b] \leq x \text{ iff } b \leq x \\ [a, b] \geq x \text{ iff } a \geq x \\ [a, b] > x \text{ iff } a > x \end{aligned} \tag{6.6}$$

The confidence interval is then calculated by first using parametric model checking to generate an algebraic expression for Φ. This expression is a multivariate rational function that depends on some or all of the unknown transitions probabilities.

Whilst the computation of this expression may be expensive it need only be computed once after which time the expression may be used many times to assess changes in the system parameters.

Example 6.2. *Consider the DTMC shown in Figure 6.2 which is adapted from Figure 5.1. Each of the transitions is now labelled with the probability of transition as a parameter and unlabelled edges have a fixed probability of 1. If we wish to assess the probability of reaching the complete state, s_6, having started in state s_0. then we can construct a PCTL formula as:*

$$\Phi = P_{=?}[F \text{ “}complete\text{”}] \tag{6.7}$$

Using the PRISM parametric model checking engine we obtain the expression:

$$\Phi = \frac{a_1 b_1}{1 + a_1 b_1 - c_1 a_2 - a_1 - a_3} \tag{6.8}$$

Having obtained an algebraic expression for Φ the second stage makes use of the transition observations to obtain confidence intervals for the expressions parameters. These are then used to derive the required confidence interval for Φ.

In order to calculate the confidence intervals associated with the state transitions the approach utilises simultaneous confidence intervals [116],[117],[118]. Assuming a state $s_i \in S$ with unknown transition probabilities then the probability of transition from state s_i to state s_j is p_{ij} and the associated confidence interval is $[\underline{p}_{ij}, \bar{p}_{ij}]$. The simultaneous $(1-\alpha_i)$ confidence intervals for the transition from s_i to each of the n states in S is then $[\underline{p}_{i1}, \bar{p}_{i1}], [\underline{p}_{i2}, \bar{p}_{i2}], \cdots [\underline{p}_{in}, \bar{p}_{in}]$..

The algebraic expression, Φ, is a function of the unknown model probabilities, as shown in (6.8), and more generally:

$$\Phi(p_{i_1 j_1}, p_{i_2 j_2}, \cdots, p_{i_m j_m}) \tag{6.9}$$

where $p_{i_1 j_1}, p_{i_2 j_2}, \cdots, p_{i_m j_m}$ are $m > 0$ unknown transition probabilities of $\mathcal{D}$.

A $(1-\alpha)$ confidence interval $[a, b]$ for Φ may be calculated and, given the memoryless property of a Markov chain, the transitions from different states s_i are independent and therefore

$$(1-\alpha) = \prod_{i \in I} (1-\alpha_i) \tag{6.10}$$

where α_i is the error level associated with the confidence interval for state s_i and $I = \{i_1, i_2, \cdots, i_m\}$. The interval limits a and b may be calculated using the following:

$$\begin{aligned} a = \quad & \text{minimize} && \Phi(p_{i_1 j_1}, p_{i_2 j_2}, \cdots, p_{i_m j_m}) \\ & \text{subject to} && \underline{p}_{i_1 j_1} < p_{i_1 j_1} < \bar{p}_{i_1 j_1} \\ & && \cdots \\ & && \underline{p}_{i_m j_m} < p_{i_m j_m} < \bar{p}_{i_m j_m} \\ & && \sum_{j=1}^{n} p_{ij} = 1 \text{ for } i \in I \end{aligned} \tag{6.11}$$

and

$$\begin{aligned} b = \quad & \text{maximize} && \Phi(p_{i_1 j_1}, p_{i_2 j_2}, \cdots, p_{i_m j_m}) \\ & \text{subject to} && \underline{p}_{i_1 j_1} < p_{i_1 j_1} < \bar{p}_{i_1 j_1} \\ & && \cdots \\ & && \underline{p}_{i_m j_m} < p_{i_m j_m} < \bar{p}_{i_m j_m} \\ & && \sum_{j=1}^{n} p_{ij} = 1 \text{ for } i \in I \end{aligned} \tag{6.12}$$

In order to solve the optimization problem above we must determine the bounds for the simultaneous confidence intervals. A number of different simultaneous confidence intervals have been proposed but, by default, FACT utilises those proposed by Kwong and Iglewicz [117] as these achieve a good trade-off between computational complexity and precision. The authors note however that this may be substituted without affecting the validity of the FACT approach.

Whilst (6.10) provides a means of calculating the confidence interval it does not suggest how the values of α_i should be selected. A naive method is to set all values to be the same i.e.

$$\alpha_i = 1 - (1 - \alpha)^{1/\#I} \tag{6.13}$$

where $\#I$ is the number of distinct intervals present in the algebraic equation. This method is suboptimal, however, and narrower confidence intervals are possible. Consider a property applied to the IT support system where the algebraic expression returned is

$$\Phi(\mathsf{a}_1, \mathsf{b}_1) = 0.9\mathsf{a}_1 + 0.0001\mathsf{b}_1 \tag{6.14}$$

and we desire a 0.95 confidence interval for Φ.

The parameters a_1 and b_1 are associated with states s_1 and s_2 respectively which have associated confidence error levels α_1 and α_2. Therefore

$$(1 - \alpha_1)(1 - \alpha_2) = 0.95 \tag{6.15}$$

and naively we would choose

$$(1 - \alpha_1) = (1 - \alpha_2) = \sqrt{0.95} = 0.975. \tag{6.16}$$

If we consider (6.14) then we see that Φ is more sensitive to changes a_1 and therefore it is desirable to have a narrower confidence interval for a_1 at the expense of increasing that associated with b_1 as this would decrease the interval associated with Φ. A better solution would therefore be

$$(1 - \alpha_2) = 0.951 \tag{6.17}$$

$$(1 - \alpha_1) = 0.95/0.951 = 0.998 \tag{6.18}$$

A similar analysis shows that our naive assumption of confidence interval symmetry is suboptimal when the number of observations associated with transitions is very different, e.g. $N_1 \gg N_2$.

Finding the optimal confidence intervals then requires an optimization procedure and a hill climbing heuristic is suggested for this purpose.

6.2 The FACT tool

To ease the adoption of formal verification with confidence intervals we developed FACT which accepts parametric Markov chains specified in an extended version of the PRISM [8] high-level modelling language. Whilst we have chosen PRISM as our parametric model checking engine of choice, improving the efficiency of parametric model checking is an active research area with new tools and techniques being produced regularly [11, 5, 115]. The theory which underpins FACT is agnostic to the choice of model checking engines and as such FACT has been architected to allow for the use of alternate model checking engines.

PRISM models, and by extension FACT, describe systems as a parallel composition of a set of modules. The state of a module is encoded by a set of finite-range local variables, and its state transitions are defined by probabilistic guarded commands that change these variables, and have the general form:

$$[action]\ guard -> e_1 : update_1 + e_2 : update_2 + \cdots + e_n : update_n \tag{6.19}$$

In this command, *guard* is a boolean expression over all model variables. If guard evaluates to true, the arithmetic expression $e_i, 1 \leq i \leq n$, gives the probability with which the $update_i$ change of the module variables occurs. When action is present then all commands with this action, across all modules, have to synchronise (i.e., to carry out one of these commands simultaneously).

FACT extends the PRISM language to allow the replacement of arithmetic expressions $e_1, e_2, \cdots, e_n$ with parameters $\mathsf{x}_1, \mathsf{x}_2, \cdots, \mathsf{x}_n$ which are specified with the FACT declaration:

$$\textbf{param double}\ \mathrm{x} = t_1\ t_2\ \cdots\ t_n \tag{6.20}$$

where $t_i \in \mathbb{N}, 1 \leq i \leq n$ is the number of transitions associated with $update_i$ that were observed from states which satisfy the *guard*.

Example 6.3. *Consider the DTMC of the IT ticket support process as shown in Figure 6.2 where the observed system transitions are those given in Table 5.2. A traditional DTMC model using point estimates could be described as:*

```
// Abstract IT ticket support model

dtmc

const double p_12 = 533/(533+114+24+34);
const double p_13 = 24/(533+114+24+34);
const double p_14 = 34/(533+114+24+34);
const double p_17 = 114/(533+114+24+34);

const double p_25 = 32/(501+32);
const double p_26 = 501/(501+32);

const double p_31 = 16/(16+8);
const double p_37 = 8/(16+8);

module main

    s: [0..7] init 0;

    [] s=0 -> (s'=1);
    [] s=1 -> p_12:(s'=2) + p_13:(s'=3) + p_14:(s'=4) + p_17:(s'=7);
    [] s=2 -> p_26:(s'=6) + p_25:(s'=5);
    [] s=3 -> p_31:(s'=1) +p_37:(s'=7);
    [] s=4 -> (s'=1);
    [] s=5 -> (s'=1);
    [] s=6 -> (s'=6);
    [] s=7 -> (s'=7);

endmodule
```

In order to rewrite this as a FACT program and utilise the observation counts we replace the constant definitions associated with transitions. For transitions leaving state s_1 we therefore write:

$$\textbf{\textit{param double}}\ x = 533\ \ 24\ \ 34\ \ 114; \tag{6.21}$$

It is then necessary to modify the guarded command in order to reference the count variables x as shown in (6.19). For state s_1 the guarded command therefore becomes:

$$[]\ s{=}1\ \ {->}\ \ x1{:}(s'{=}2) + x2{:}(s'{=}3) + x3{:}(s'{=}4) + (1{-}x2{-}x3){:}(s'{=}7); \tag{6.22}$$

where x1 is calculated with reference to the first observation count for the variable x and represents the probability of transitioning to state s_2. x2 and x3 are then calculated with reference to the second and third observation count respectively.

A complete FACT model for the IT support system is then written as follows:

```
// FACT model for IT ticket support system
dtmc

param double x = 533 114 24 34;
param double y  = 32 501;
param double z = 16 8;

module main

    s: [0..7] init 0;

    [] s=0 -> (s'=1);
    [] s=1 -> x1:(s'=2) + x2:(s'=3) + x3:(s'=4) + (1-x1-x2-x3):(s'=7);
    [] s=2 -> y1:(s'=6) + (1-y1):(s'=5);
    [] s=3 -> z1:(s'=1) + (1-z1):(s'=7);
    [] s=4 -> (s'=1);
    [] s=5 -> (s'=1);
    [] s=6 -> (s'=6);
    [] s=7 -> (s'=7);

endmodule
```

FACT has a modular architecture as shown in Figure 6.3. Each step of the process is then carried out by a different module. The tool was written in Java and provides a graphical user interface for users to enter models and PCTL properties for analysis as well as evaluation parameters.

The user provides a description of the system using the FACT modelling language and a PCTL property for analysis. When the property is selected for evaluation the user is asked to enter a range of confidence levels over which to calculate the property value. The user may also optionally, choose to use the hill climbing heuristic to reduce the width of the confidence interval. Given these inputs the verification manager generates a confidence interval for each confidence level α in a four step process.

For the parametric quantitative verification engine FACT utilises PRISM which is called to generate the algebraic equation. This is time consuming but is only called once. This module could be replaced by alternate modelling checking engines such as PARAM [115] or STORM [11].

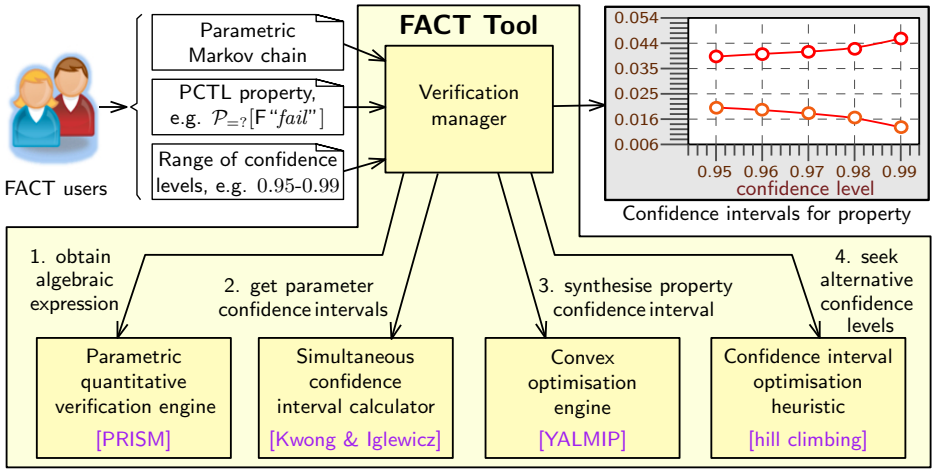


Figure 6.3: Estimating the mean of a stochastic process using sample sets of varying sizes.

Simultaneous confidence intervals are constructed using the solution proposed by Kwon and Iglewicz as proposed in the theoretical work by Calinescu et al. although several alternative calculations are available as described in [23] and could be substituted if appropriate.

The confidence interval synthesis utilises MATLAB and the YALMIP optimization toolbox. Since MATLAB is a commercial product it may be worth investigating alternate open source optimization tool kits using frameworks such as Octave (http://www.gnu.org/software/octave).

Finally, if the hill climbing optimization heuristic has been selected, this is called. Whilst FACT implements the heuristic proposed in [23] there are numerous such optimization heuristics which could be substituted for this module.

6.3 Using the FACT tool

In order to demonstrate the FACT tool we consider a business-critical web application taken from [23] which comprising an HTTP proxy server, a web server and an application server. To serve client requests, the web application accesses structured data and static content, for example text files and images, stored in a database and on a file server, respectively. Both types of static content are cached by ad-hoc cache servers.

The parametric Markov chain shown in Figure 6.4 models the functionality that handles an HTTP request within the web application. Each state represents a stage of the handling process. The initial state s_1 corresponds to the request being received, and the shaded states are absorbing states, i.e., states that once entered cannot be left. These states indicate the outcome of the request handling, i.e., whether the request handling succeeds (s_9) or fails due to an unavailable server (s_8) or to the overloading of a component of the application (s_{10}).

When the process transitions from states s_1, s_4, s_6 and s_7 these are recorded in a log and the number of observed transitions are listed in Table 6.2. These transition counts were generated through simulation using a model with fixed probabilities. These probabilities are then assumed unknown in the analysis phase. All other transition probabilities are obtained as point estimates from domain experts.

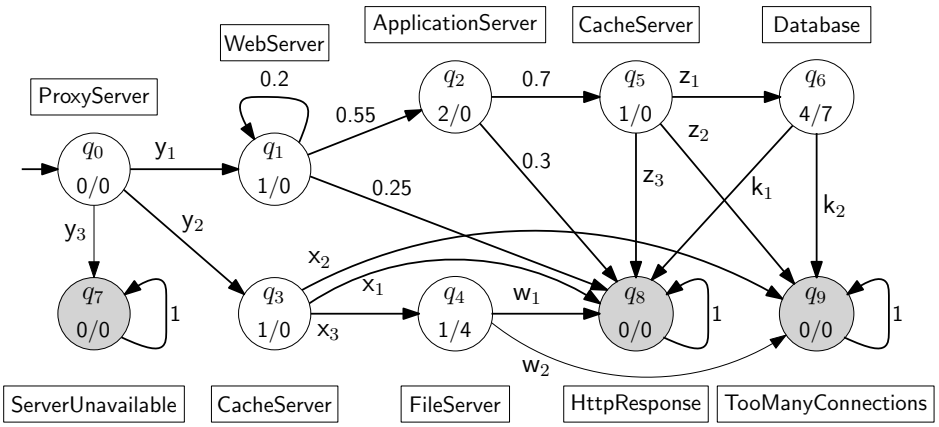


Figure 6.4: PMC modelling the handling of an HTTP request process taken from [23].

Table 6.2: Web application observation counts

from	to	parameter	$O^{\#}$	Point Estimate
s_1	s_2	y_1	4050	0.4054
s_1	s_4	y_2	5938	0.5944
s_1	s_8	y_3	2	0.0002
s_4	s_{10}	x_1	5723	0.5622
s_4	s_9	x_2	4	0.0004
s_4	s_5	x_3	4452	0.4374
s_5	s_9	w_1	9784	0.9995
s_5	s_{10}	w_2	4	0.0005
s_6	s_7	z_1	2467	0.2499
s_6	s_{10}	z_2	10	0.0010
s_6	s_9	z_3	7395	0.7491
s_7	s_9	k_1	9964	0.9964
s_7	s_{10}	k_2	6	0.0006

A FACT model is created with five parameters and their associated observation counts. A full listing of the model is given in Listing 6.1. Each of the non-absorbing states in the model are annotated with costs c/t where c is the average cost (in tenths of a cent) of entering that state and t is the average time (ms) the process is expected to spend in that state.

Table 6.3 shows the Quality-of-Service (QoS) properties that we will assess for the process with an informal description and PCTL formula for each.

```
probabilistic

param double y = 4050 5938 2;
param double x = 5723 4 4452;
param double w = 9784 4;
param double z = 2467 10 7395;
param double k = 9964 6;

module M1
    q : [0..9] init 0;

    [] q=0 -> y1:(q'=1) + y2:(q'=3) + (1-y1-y2):(q'=7);
    [] q=1 -> 0.2:(q'=1) + 0.55:(q'=2) + 0.25:(q'=8);
    [] q=2 -> 0.7:(q'=5) + 0.3:(q'=8);
    [] q=3 -> x1:(q'=8) + x2:(q'=9) + (1-x1-x2):(q'=4);
    [] q=4 -> w1:(q'=8) + (1-w1):(q'=9);
    [] q=5 -> z1:(q'=6) + z2:(q'=9) + (1-z1-z2):(q'=8);
    [] q=6 -> k1:(q'=8) + (1-k1):(q'=9);
    [] q=7 -> 1:(q'=7);
    [] q=8 -> 1:(q'=8);
    [] q=9 -> 1:(q'=9);
endmodule

rewards "cost"
    q=1 : 1;
    q=2 : 2;
    q=3 : 1;
    q=4 : 1;
    q=5 : 1;
    q=6 : 4;
endrewards

rewards "time"
    q=4 : 4;
    q=6 : 7;
endrewards
```

Listing 6.1: FACT model for a web application

The model and properties are then entered in to the FACT tool as shown in Figure 6.5 and FACT is used to generate confidence intervals for each of the properties with confidence levels in the range $85 - 99\%$. The resulting graphs are shown in Figure 6.6 and, for each confidence level, an upper and lower bound on the interval is displayed such that any value outside of this interval may be interpreted as violating the property at the stated confidence level. For all of the results returned we see that the interval widens as the confidence level increases, as expected.

Table 6.3: PCTL properties for the web application process

ID	Formula	Description
P1	Reliability: What is the probability of successfully handling a request?	$\mathcal{P}_{=?}[\mathrm{F}\ \mathsf{HttpResponse}]$
P2	Cache hit probability: What is the probability that requests are handled without accessing the database or the file server?	$\mathcal{P}_{=?}[\neg(\mathsf{Database} \vee \mathsf{FileServer})\mathrm{U}\ \mathsf{HttpResponse}]$
R1	Cost: What is the expected cost for handling a request ?	$\mathcal{R}^{\mathsf{cost}}_{=?}[\mathrm{F}\ \mathsf{Done}]$
R2	Response time: What is the expected response time?	$\mathcal{R}^{\mathsf{time}}_{=?}[\mathrm{F}\ \mathsf{Done}]$

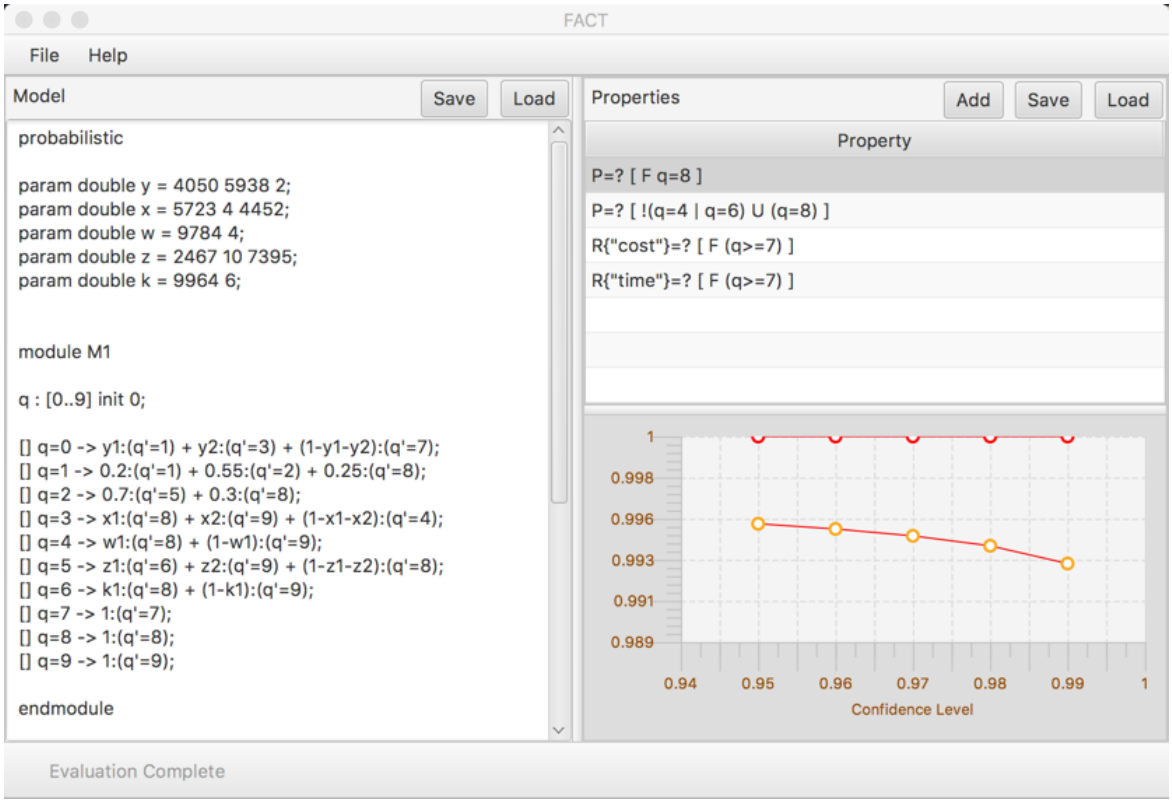


Figure 6.5: Screenshot of the FACT application for the HTTP request process

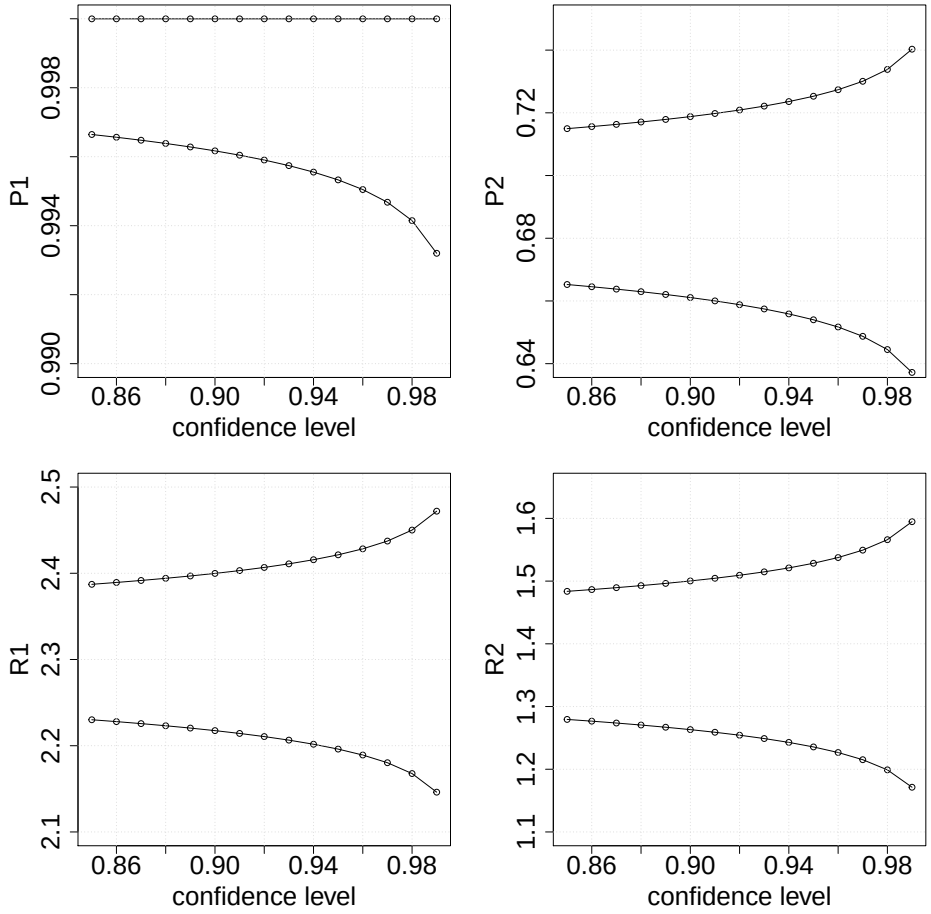
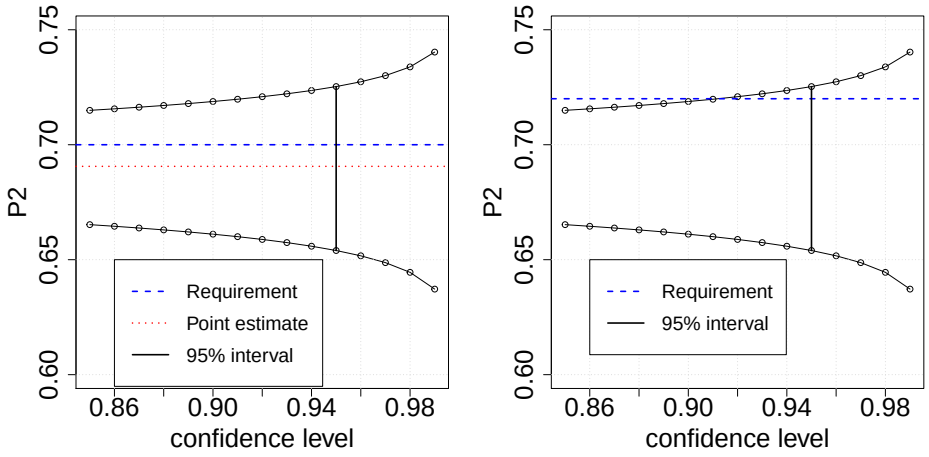


Figure 6.6: Confidence intervals for HTTP request process

Let us consider then a requirement that the cache hit probability is at least 0.7 with a confidence level of 0.95. When the PMC is constructed using point estimates, and evaluated using PRISM, the value returned is 0.6906 which indicates that the designer would reject the process as it falls below the 0.7 threshold. Verification with confidence intervals however shows that the value of 0.7 falls within the confidence bounds for all confidence levels between 0.85 and 0.99. This is shown in Figure 6.7a. Rejecting this process on the basis of the observation evidence would therefore be incorrect.

Choosing a confidence level at which to evaluate a property is not always straight forward however. Consider the case where the requirement is for a cache hit probability of at least 0.72. This is shown in figure 6.7b. With a confidence level of 0.90 the property is violated however we can not say that the property is violated at the 0.95 confidence level as 0.72 lies within the confidence bounds.



(a) Comparison to Point estimates (b) Selecting confidence levels

Figure 6.7: Interpreting FACT verification results.

6.4 Evaluation

To evaluate FACT we applied the tool to five systems from a range of application domains. For each system we synthesise confidence intervals for PCTL-encoded reliability, performance and cost properties of parametric Markov chains. Table 6.4 summarises the experimental results obtained for the PMCs of:

- a web application, shown in Section 6.3, and taken from [23] (Web);
- a tele-assistance service-based system adapted from [69, 105] (TAS);
- the low-power wireless bus communication protocol taken from [23] (LWB);
- the bounded retransmission protocol from the PROPhESY [119] site (BRP);
- the Zeroconf IP address selection protocol from the PARAM [115] website (Z).

The timing results were obtained on a standard OS X 10.8.5 MacBook computer with 1.3GHz Intel Core i5 processor and 8GB 1600MHz DDR3 RAM. The models, PCTL property files, results and descriptions for all case studies are available on our FACT website `http://www-users.cs.york.ac.uk/~cap/FACT`.

These case studies demonstrated several key benefits of our probabilistic model checker. First, FACT supports the analysis of systems for which state transition probabilities are unknown, but observations of these transitions are available from logs or run-time monitoring. Second, it enables the analysis of reliability, performance and other QoS properties of systems at the required confidence level. Third, it can prevent invalid design and verification decisions (as shown in Figure 6.7). In many scenarios, the quantitative analysis of Markov models built using point estimates of the unknown transition probabilities misleadingly suggested that requirements were met. In contrast, FACT showed that this was only the case with low confidence levels that are typically deemed unacceptable in practice. Last but not least, our case studies showed that FACT can be used to analyse systems from multiple domains.

Table 6.4: Experimental results for the case studies from Section 6.4

PMC	$psets$[a]	$params$[b]	PCTL property	t^{c}_{exp}	t^{d}_{CI}
Web	5	13	$\mathcal{P}_{=?}$[F HttpResponse]	0.75s	3.96s
			$\mathcal{P}_{=?}$[¬(Database ∨ FileServer) U HttpResponse]	0.84s	3.43s
			$\mathcal{R}^{\text{cost}}_{=?}$[F Done]	0.86s	3.31s
			$\mathcal{R}^{\text{time}}_{=?}$[F Done]	0.89s	3.29s
TAS	3	6	$\mathcal{P}_{=?}$[F FailedAlarm]	0.24s	4.32s
			$\mathcal{P}_{=?}$[¬Done U FailedService]	0.12s	2.82s
			$\mathcal{P}_{=?}$[¬Done U FailedAlarm{MedicalAnalysis}]	0.11s	2.78s
LWB	1	2	$\mathcal{R}^{\text{power}}_{=?}$[S]	0.24s	3.03s
			$\mathcal{R}^{\text{energy}}_{=?}$[F StartedUp]	0.27s	2.98s
BRP	2	4	$\mathcal{P}_{=?}$[F SenderNoSuccessReport]	0.44s	31.6s
Z	2	4	$\mathcal{R}^{\text{numTests}}_{=?}$[F DecisionMade]	0.15s	5.41s

[a]number of parameter sets (6.20) in the PMC [b]total number of PMC parameters
[c]time to compute algebraic expression [d]time to synthesise confidence interval

Whilst FACT has been shown to be applicable in a range of contexts the models considered and the number of parameters are relatively small. As models become more complex parametric model checking becomes infeasible and increasing the scalability of parametric model checking remains an active research area [5].

In Chapter 7 we show how the concepts which underpin FACT, and the tool itself, can be used to compare models with reference to observed observation logs in order to identify normal and abnormal behaviours in operational processes. We then evaluate the application of FACT within this context in chapter 8.

6.5 Related work

Deriving DTMC models from observation data is a well known statistical problem [106],[120] and has become a common approach for deriving DTMC models of user behaviour [70],[121]. Unlike FACT, however, this work assumes that a single point estimate is sufficient to characterise the parameters of the model under consideration.

These techniques, and those proposed by FACT, assume that the observations describe parameters which, whilst stochastic in nature, do not change over time. A parallel stream of research concerns the identification of parameters at run-time such that changes in parameters values can be reflected in the models used for verification. Zheng et al. [122] describe how Kalman filters can be used to update performance models continuously through observed data streams. Epifani et al. [123],[124] utilise a Bayesian estimator to update models at run time to ensure that the performance models more accurately reflect the current state of the system under review. Calinescu et al. [125],[105] use a technique of observation ageing in which older observations are weighted to reduce their effect on current parameter estimates. These techniques are complimentary to the FACT approach and extending FACT to allow for run-time adaptation in this presents an interesting avenue for future work.

Whilst interval-valued discrete time Markov chains (IDTMC) have previously been proposed [126],[127] and extended to allow for PCTL verification with uncertainty [128],[129] FACT still offers several unique benefits. Firstly FACT operates with parametric Markov chains in which transition probabilities are specified as observation counts and as such this makes FACT particularly useful for practical applications in which transition probabilities are unknown but log data is available. Secondly FACT is able to establish confidence intervals for PCTL at any requested confidence level. Thirdly the hill-climbing optimization technique used in FACT is able to reduce the width of the confidence interval produced over a number of iterations. Finally FACT can be readily used due to the availability of a freely downloadable model checking tool.

FACT relies on parametric model checking as a key component of the analysis. Whilst parametric model checking tools has matured over recent years, with support offered by the leading model checkers PARAM [115], PRISM [8] and Storm [11], the computational cost of parametric model checking remains high and, as such, the range of models analysable by these tools remains limited. In recent work [5] we show how reductions in model checking times of several orders of magnitude can be achieved for service based systems and envisage that other domains could benefit similarly. The modular nature of the FACT architecture allows for the current PMC of choice (PRISM) to be replaced as new model checking tools are developed.

An alternative approach to evaluating Markovian models with parameter uncertainty is to use statistical model checking [130]. Unlike the numerical algorithms used for model checking in FACT which require certain model structures to hold, e.g. Markovian, statistical approaches allow these restrictions to be relaxed. Furthermore statistical approaches allow for the verification of much larger systems. Just as PARAM, PRISM and Storm have been developed to implement numerical model checking algorithms, tools which allow for statistical model checking have been developed including YMER [131], VESTA [132] and MRMC [133].

Statistical techniques utilise a simulation-based approach [134]. The key idea is to observe the system over many sample executions and then draw conclusions about the performance of the system with respect to the system requirements. Work by Meedeniya et al. [88] demonstrated how statistical model checking could be used to evaluate probabilistic models with uncertain model parameters. Statistical methods have low memory requirements and are applicable to a wider range of system modelling paradigms than numerical techniques however, they are computationally expensive when high accuracy is required. By contrast numerical methods allow for highly accurate results but the computational expense increases as the number of states required to model a system grows. Properties analysed with statistical model checking tools are typically given as a confidence interval but, unlike FACT, this interval describes the uncertainty associated with the simulation process itself rather than a confidence interval based on the level of uncertainty associated with model parameters. Indeed, a confidence interval is generated even when all parameters are fixed and known precisely.

Chapter 7

Detecting Abnormal User Behaviour using Observation-Enhanced DTMC Analysis

Using observation data to improve the accuracy of DTMC model analysis relies on the assumption that all entries in the trace data are identically distributed. Consider a task for which two options are available. The first option is costly but safe while the second is less costly but incurs a greater level of risk. The probability of a user adopting option one can be considered to be a function of the user's risk profile. As such aggregating user behaviours results in a probability which may be inaccurate for all users. Conversely, knowing which user is involved in the process would allow for a more accurate assessment of the model parameters associated with the task of interest. Indeed Ghezzi et al. [70] suggest that by partitioning log data, using information about users such as geographical location, and constructing models for each partition it is possible to compare the behaviour of users.

In this chapter we consider operational processes which are enacted by a strictly controlled set of authenticated system users. These users are entrusted with the correct operation of the process and, where their behaviour is abnormal, the effect on the organisation can be detrimental. However, identifying users who are displaying abnormal behaviours through the partitioning logs is not a simple task. We do not know in advance which users, if any, are acting abnormally and indeed identifying normal behaviour in a human-centric process can be challenging.

If we are able to identify abnormal behaviour the potential rewards are great since security incidents caused by abnormal user behaviour may result in severe financial and reputational loss [135],[136]. Insider threats arise when trusted users of an information system can exploit their access permissions to compromise the confidentiality, integrity or availability of an organisation's information assets [137],[138]. These trusted users include employees, contractors and business partners who can cause harm intentionally (e.g. for personal gain or revenge) or through error (e.g. due to negligence or insufficient training) [139].

To mitigate these threats, information systems may employ control mechanisms that restrict access to their assets. More often than not, these mechanisms implement the *role-based access control* (RBAC) [140] model, where the permissions to execute operations on information assets are associated with *roles*, and the users are only assigned the role(s) they need to perform their jobs. As such, RBAC restricts user access, and helps detect access violation attempts. However, it cannot detect users who maliciously or accidentally abuse their legitimate access permissions. Furthermore, it is unable to mitigate such abuse [141], even when a separate insider attack detection mechanism [142] is available. For example, in the context of a ticket support system, a support attendant with an elevated number of tickets opened on behalf of clients that are abandoned can be considered an anomaly not detectable by RBAC.

We propose an approach which addresses this limitation of traditional RBAC in the context of business processes. To this end, we exploit activity logs already available for many important businesses processes, which also enable the monitoring of the activities undertaken by individual users of these processes. Using this information, dynamic access control mechanisms can be employed to respond to abnormal user behaviour through actions decided based on risk analysis and pre-defined *adaptation policies*. Such actions may include changes to authorisation policies, modifications of user assignment to roles and of role permissions, user training, and changes to the business process.

We introduce a self-adaptive RBAC (saRBAC) approach that enacts these general principles by dynamically reconfiguring user assignments to roles in order to mitigate insider threats. As an example, users with abnormal behaviour may be removed from a role or may be demoted to roles with restricted permissions. Our saRBAC approach is underpinned by observation-enhanced DTMC analysis that enables the comparison of individual user behaviour to the average behaviour of the other users in the same role. Given a business process and traces of its execution obtained through monitoring, saRBAC (a) builds a parametric Markov model of the process, and (b) uses FACT to establish confidence intervals for model properties associated with key aspects of user behaviour. For each user and analysed property, two confidence intervals are computed corresponding to the property value for the user, and for all the other users taken together, respectively. If the two confidence intervals do not overlap, then saRBAC concludes that the examined user behaves (statistically) differently from the other users. The analysed properties, the definition of what constitutes abnormal behaviour, and the actions required when such behaviour is detected are formally specified in saRBAC adaptation policies.

The rest of this chapter is organised as follows. In Section 7.1 we present a real-world case study which will serve as our motivating example and aid in evaluating our approach. Section 7.2 then describes the saRBAC approach and the formulation of formalised policies for analysing user behaviour and adaptation of user access. Finally, Section 7.3 compares our approach with related research.

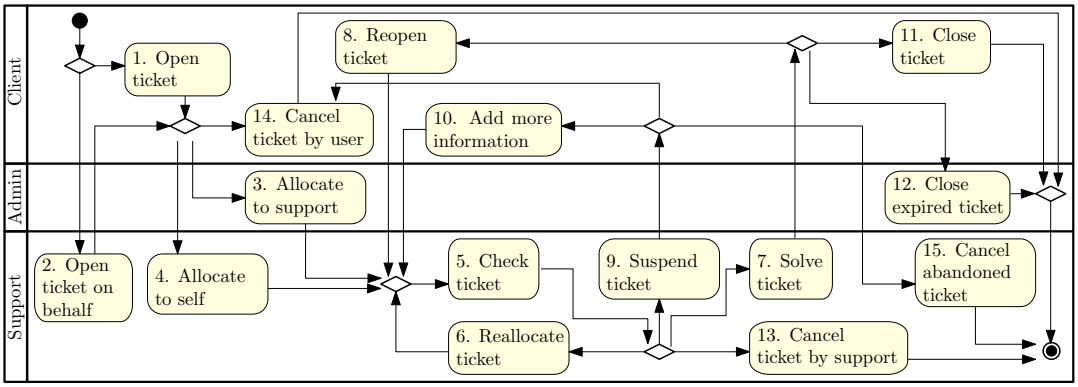


Figure 7.1: Ticket support business process (UML activity diagram produced during IFRN's development of the SUAP information system)

7.1 Motivating example

We motivate and evaluate our abnormality detection approach using the IT support system first introduced in Section 5.2.1. The IT support system is part of SUAP[1], the information system of the Federal Institute of Education, Science, and Technology of Rio Grande do Norte (IFRN), Brazil. The IFRN SUAP users comprise 12,500 academic, administrative, technical staff and contractors, and 32,000 students, located at the 26 IFRN campuses in the state of Rio Grande do Norte.

In this work we once more consider the *Ticket Support* process and a UML activity diagram for this process is provided in Figure 7.1. This process allows IFRN users to request IT support services (e.g. access and password changes) and report IT-related problems. Many of SUAP's business processes, including IT support, are security sensitive, and the RBAC access control model is used to enforce security policies.

In this context, the Ticket Support process involves three roles: *Client*, *Support* and *Administrator*. A *Client* is any user who needs an IT related service performed on their behalf. They can raise issues by opening tickets and subsequently accept the work done by closing the ticket once marked as solved. A *Support* user is an employee responsible for dealing with tickets, e.g. an IT technician or analyst. The system also includes the *Administrator* role, which comprises users responsible for supervising the work carried out by *Support* users.

As shown in Figure 7.1 the Ticket Support process starts with the opening of a ticket, either by a client (*Open ticket*) or by a support attendant on behalf of a client (*Open ticket of behalf*), e.g., the client might go to the IT department in person to raise an issue. An open ticket can be cancelled by the client (*Cancel ticket by user*), e.g., if the ticket was opened by mistake, or can be allocated to a support attendant. Ticket allocation can be done by an administrator (*Allocate to support*), or by support attendants themselves (*Allocate to self*).

Once allocated, a support attendant will work on the ticket (*Check ticket*)—solving the issue (*Solve ticket*), reallocating the ticket to a different support attendant (*Reallocate ticket*), cancelling the ticket (*Cancel ticket by support*), or suspending the ticket and asking the

[1] *Sistema Unificado de Administrao Pública* – Unified System for Public Administration

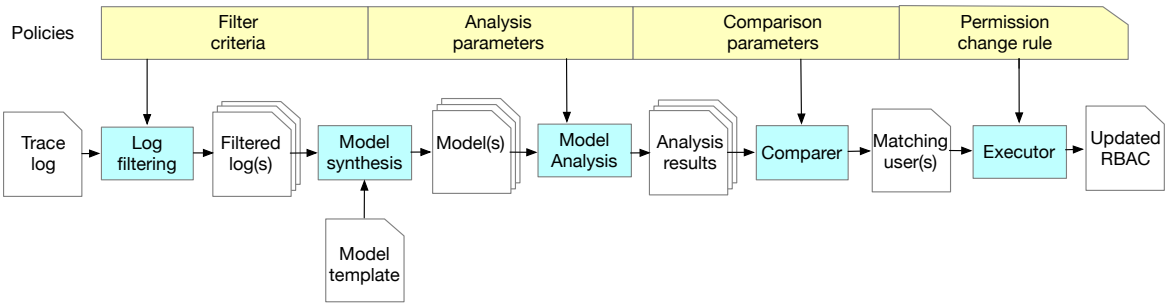


Figure 7.2: A high level diagram of the saRBAC approach

client to provide more information about the issue (*Suspend ticket*). Suspended tickets are sent back to clients, who can either reply to the support attendant (*Add more information*), or cancel the ticket (*Cancel ticket by user*). If the client does not reply within a specific time, the ticket is considered abandoned and is cancelled by the support attendant (*Cancel abandoned ticket*).

A solved ticket is sent back to its client, who can confirm the resolution of the issue by closing it (*Close ticket*), or reopen the ticket (*Reopen ticket*) indicating that the issue is not resolved. A solved ticket not handled by the client within a certain time frame is closed by an administrator (*Close expired ticket*).

Our collaborators at IFRN used SUAP's extensive logging capabilities to obtain detailed execution traces of the Ticket Support process for the three-month period between May–July 2016, and interviewed the IFRN management team to determine the organisation's concerns for the process. As such, we learnt about concerns based both on past cases of internal abuse and on yet unconfirmed incident scenarios identified by their security risk management procedures.

Using these concerns, we defined a set of adaptation policies that capture abnormal behaviours of Ticket Support users, and preferred ways of dealing with them. Table 7.1 shows a representative subset of these policies, expressed informally for measurable attributes of user behaviour. The informal and ambiguous nature of these policies is typical of requirements specified in natural language [143] and highlights the need for a more formal policy specification.

7.2 The saRBAC approach

Our five stage approach, shown diagrammatically in Figure 7.2, is applicable to business processes where a description of the process is available as a UML activity diagram such as that provided in Figure 7.1 and a set of policies, such as those shown in Table 7.1, have been defined.

As each individual interacts with the system they will leave entries in a trace log which records information about the activity undertaken by the user. In order to evaluate the behaviour of system users, with respect to the informal policy set, it is necessary for us to

Table 7.1: Adaptation policies for the IT support system

ID	Description
P1	A client whose expected number of reopens per ticket is larger than that of the other clients should be not be allowed to open new tickets.
P2	A support attendant whose expected number of suspensions per ticket exceeds that of the other attendants should be closely monitored and should need approval for suspending tickets. Additionally, the clients that have been served by this attendant should be considered; if their expected number of suspensions per ticket *for tickets handled by other attendants* is lower than that of the other clients, then the investigated attendant should be suspended.
P3	A support attendant whose probability of cancelling ticket that have not been suspended exceeds that of the other support team members should need approval for cancelling tickets.
P4	A support attendant whose expected number of tickets opened on behalf of clients are abandoned exceeds that of the other support team members should no longer be allowed to open tickets on behalf of clients.
P5	A support attendant whose expected number of reallocations, suspensions, cancellations or reopens per ticket exceeds that of the other support attendants should be placed under observation. Additionally, if this discrepancy is also confirmed at a higher confidence level then the support attendant should need approval for his or her actions.
P6	A client whose expected number of suspensions, reopens, abandonments or cancellations by support per ticket exceeds that of the other clients should be placed under observation. Additionally, the support attendants that have dealt with by this client should be considered; if their expected number of suspensions, reopens, abandonments or cancellations by support per ticket is lower than that of the other support attendants, then the investigated client should not be allowed to open new tickets.

describe these policies in a more formal manner where the resultant policy is composed of four elements:

- Filter criteria: Used to control the partitioning and filtering of the trace log to examine user behaviours.
- Analysis parameters: Used by a model analysis engine to exercise the models and obtain a result set which characterises the user behaviour.
- Comparison parameters: Used to compare results of model analysis and assess whether users have violated thresholds which will lead to action having to be taken.
- Permission change rule: The action to take to modify the user's permissions if a violation is detected.

The first stage of our approach is to filter and partition the trace log such that the resultant trace sets contain traces associated with individuals or groups of users. The second stage, model synthesis, makes use of a parametric model template and derives model parameters through an analysis of the filtered logs from the previous stage. In this way the resultant models characterise the behaviour of users associated with each trace sub set. In the third stage the models are passed to an analysis engine along with analysis parameters from the formal policy. The results generated represent a quantitative evaluation of user behaviours. In the forth stage we compare these results using parameters specified in the policy. This

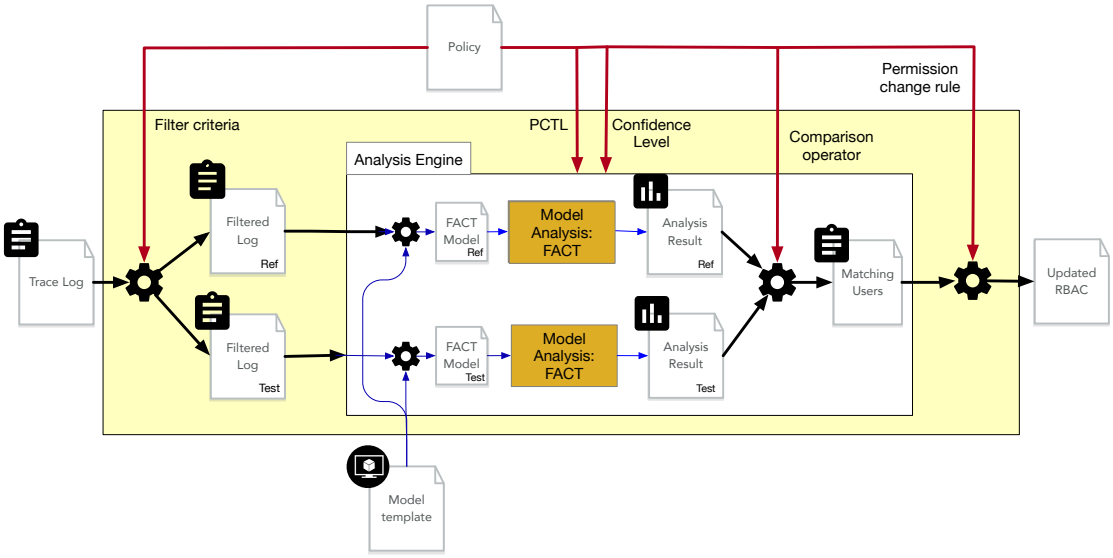


Figure 7.3: saRBAC architecture utilising FACT

comparison identifies those users that have violated the policy and returns a list of users exhibiting abnormal behaviours. The final stage takes a permission change rule from the policy and applies this to the role based access control system to modify the user permissions for those users identified in the previous stage.

In order to demonstrate how the approach may be implemented in practice we consider the motivating example given in Section 7.1 when the FACT probabilistic model checker described in Chapter 6 is used as the analysis engine of choice. Figure 7.3 then shows a high level implementation architecture of the saRBAC approach using FACT.

The trace log is first partitioned into two sub logs using the filter criteria. These sub logs are labelled "test" and "reference". The test sub log contains those traces associated with users that we wish to examine for abnormality, and the reference sub log contains all other users.

Having obtained our sub logs through the partitioning process we then undertake model synthesis by counting transitions observed in each sub log and using these as values for parameters in our model template. The template used is a FACT model with place-holder variables for the observation counts. The result is then two FACT models which correspond to the "test" and "reference" sub logs.

Next, model analysis is undertaken by the FACT engine described in Section 6.2 which requires a confidence level and a PCTL formula to encode our natural language analysis requirement. The analysis results generated for the saRBAC instantiation based on FACT are then intervals.

These intervals are then compared using an operator derived from the natural language requirement. When the comparer returns true the list of users associated with the test sub log is returned otherwise an empty list is generated.

Finally the permission change rule is used to update the access permission of any users

identified in the previous stage.

In the remainder of this section we describe the role based nature of the systems to which our approach applies before defining how log data must be structured to enable trace analysis within our approach. Finally we show how the policies from Table 7.1 can be encoded with examples based on the FACT implementation applied to the motivating example.

7.2.1 Users and roles

Our approach considers systems whose users are organised into $n > 0$ roles such that:

$$Role_1, Role_2, \cdots, Role_n \tag{7.1}$$

are the sets of users associated with each role and the set of all users is

$$Users = \bigcup_{i=1}^{n} Role_i. \tag{7.2}$$

The complete set of p distinct activities in the process is then defined as $A = \{\alpha_1, \alpha_2, \cdots, \alpha_p\}$ and we define $A_i \subset A$ to be the set of activities which a user $u \in Role_i,\ 1 \leq i \leq n$ may perform. Accordingly the set of activities that a generic user $u \in Users$ has permissions to execute is:

$$perm(u) = \{\alpha \in A \mid \exists\ 1 \leq i \leq n \bullet u \in Role_i \wedge \alpha \in A_i\} \tag{7.3}$$

7.2.2 Log data

In order to allow for the implementation of saRBAC the business process must be instrumented to capture a log of process instantiations such that the complete log $\mathcal{L}$ is a set of traces:

$$\mathcal{L} = \{T_1, T_2, \cdots, T_m\} \tag{7.4}$$

where m is the number of observed instantiations of the business process.

A trace T_i comprises a unique identifier id (for an instantiation of the process) and a sequence of events (or log entries) $E = \langle e_{i1}, e_{i2}, \ldots, e_{iN_i} \rangle$:

$$T_i = (id, E), \tag{7.5}$$

where $1 < i < m$ and N_i is the number of events in a trace sequence. Each event e is an object with q attributes $B = (\beta_1, \beta_2, \cdots, \beta_q)$ that characterise features of the activity undertaken as part of the process. As a minimum each event must contain the following attributes:

$e.user$	is the user from $Users$
$e.activity$	is an activity from A
$e.time$	is the time at which the activity was completed.

Additional attributes, such as location or priority, may also be present in the log and used to filter log data and this is discussed in the next section.

Example 7.1. *The Ticket Support process from Section 7.1 uses $n=3$ roles, i.e.*

$$\begin{aligned} Role_1 &= Client, \\ Role_2 &= Support, \\ Role_3 &= Admin. \end{aligned}$$

The activity sets for these roles are

$$\begin{aligned} A_1 &= \{\mathsf{Open}, \mathsf{Reopen}, \mathsf{AddInformation}, \mathsf{Close}, \mathsf{CancelByUser}\}, \\ A_2 &= \{\mathsf{OpenOnBehalf}, \mathsf{AllocateToSelf}, \mathsf{Check}, \mathsf{Reallocate}, \mathsf{Solve}, \\ &\quad \mathsf{Suspend}, \mathsf{CancelBySupport}, \mathsf{CancelAbandoned}\}, \\ A_3 &= \{\mathsf{AllocateToSupport}, \mathsf{CloseExpired}\}, \end{aligned}$$

and the entire activity set is $A = A_1 \cup A_2 \cup A_3$.

A client user $u_1 \in Client$ starts a process by opening a ticket which is allocated a unique identifier of 1. Support user, $u_2 \in Support$, decides they can service the ticket without involving other members of the support team and therefore allocates it to themself. They check the ticket and undertake the work required to service the request before marking it as solved. Finally user u_1 sees the work has been completed and decides that the ticket can be closed.

The trace is added to the complete log $\mathcal{L}$ as $T = (1, E)$ where E contains 5 distinct events $E = \langle e_1, e_2, e_3, e_4, e_5 \rangle$ and:

$$\begin{array}{lll} e_1.user = u_1 & e_1.activity = \mathsf{Open} & e_1.time = 09:00 \\ e_2.user = u_2 & e_2.activity = \mathsf{AllocateToSelf} & e_1.time = 09:12 \\ e_3.user = u_2 & e_3.activity = \mathsf{Check} & e_1.time = 09:13 \\ e_4.user = u_2 & e_4.activity = \mathsf{Solve} & e_1.time = 09:48 \\ e_5.user = u_1 & e_5.activity = \mathsf{Close} & e_1.time = 10:04 \end{array}$$

7.2.3 Step1 : Partitioning and filtering trace data

The first step of the approach is to partition and filter the data log with reference to rules and constraints which encode the intention of the informal policy. These rules define the scope of the policy which comprises combinations (i.e., tuples) of $r \geq 1$ subsets of traces.

Given a generic trace set $\mathcal{L}$, the complete scope of a policy is:

$$scope(\mathcal{L}) = \big\{(T_{label_1}, T_{label_2}, \ldots, T_{label_r}) \in (\mathbb{P}\mathcal{L})^r \mid rule_1 \wedge rule_2 \wedge \ldots\big\}, \tag{7.6}$$

where $\mathbb{P}\mathcal{L}$ denotes the powerset (i.e., the set of all subsets) of $\mathcal{L}$ and $rule_1, rule_2, \ldots$ are constraints over the elements of traces from $\mathcal{L}$, expressed in first order logic. Note that labels $label_1, label_2, \ldots, label_r$ are associated with the r subsets of traces from each combination to

enable the other policy components to refer to these subsets easily.

Example 7.1. *Let us consider Policy P1 from Table 7.1. This policy requires the log data to be partitioned into two sub logs for every client user $u \in Client$. The first sub log contains those traces associated with tickets in which the client of interest has been involved. The second contains the traces of all other tickets.*

For each client $u \in Client$, a partition (T_{test}, T_{ref}) of the trace data may be constructed such that

$$\begin{aligned}\big(T_{test} = \{(id, \langle e_1, e_2, \dots e_N\rangle) \in \mathcal{L} \mid \exists 1 \leq j \leq N \bullet e_j.user = u\} \wedge \\ T_{ref} = \mathcal{L} \setminus T_{test}\big)\end{aligned} \tag{7.7}$$

The resulting partition is labelled such that T_{test} contains those traces associated with a client user u and T_{ref} is the reference set against which u will be compared.

The complete scope of this policy is then the following set of (T_{test}, T_{ref}) combinations:

$$\begin{aligned}scope(\mathcal{L}) = \big\{(T_{test}, T_{ref}) \in \mathbb{P}\mathcal{L} \times \mathbb{P}\mathcal{L} \mid \exists u \in Client \bullet \\ \big(T_{test} = \{(id, \langle e_1, e_2, \dots e_N\rangle) \in \mathcal{L} \mid \exists 1 \leq j \leq N \bullet e_j.user = u\} \wedge \\ T_{ref} = \mathcal{L} \setminus T_{test}\big)\big\}\end{aligned} \tag{7.8}$$

This partitions the entire log $\mathcal{L}$ into pairs of sets without any filtering, i.e., $T_{test} \cup T_{ref} = \mathcal{L}$ and $T_{test} \cap T_{ref} = \emptyset$.

Let us now consider a policy which is only concerned with those tickets created after a time τ. Since the ticket creation event is always the first in a trace sequence we can add a single rule and our complete policy rule becomes:

$$\begin{aligned}scope(\mathcal{L}) = \big\{(T_{test}, T_{ref}) \in \mathbb{P}\mathcal{L} \times \mathbb{P}\mathcal{L} \mid \exists u \in Client \bullet \\ \big(T_{test} = \{(id, \langle e_1, e_2, \dots e_N\rangle) \in \mathcal{L} \mid e_1.time \geq \tau \wedge \exists 1 \leq j \leq N \bullet e_j.user = u\} \wedge \\ T_{ref} = \{(id, \langle e_1, e_2, \dots e_N\rangle) \in \mathcal{L} \mid e_1.time \geq \tau\} \setminus T_{test}\big)\big\}\end{aligned} \tag{7.9}$$

7.2.4 Step 2 : Model synthesis

Model synthesis requires a model template in which place-holder variables are replaced by values derived from an analysis of filtered logs. A function, $\mathcal{F}^m$, is applied to each partition of the filtered log data using the model template, M_T, to generate the r models necessary to evaluate the policy requirement for that log partition.

For each partition $(T_{label_1}, T_{label_2}, \dots, T_{label_r}) \in scope(\mathcal{L})$

$$\mathcal{F}^m((T_{label_1}, T_{label_2}, \dots, T_{label_r}), M_T) = (M_1, M_2, \cdots, M_r)$$

In general $\mathcal{F}^m$ will be constructed with knowledge of the analysis engine to be utilised and the nature of the parameters present in the template model.

Example 7.2. *For policy P1 the log data is partitioned into trace sets which characterise the behaviour of users from the Client role. With the remaining data characterising the*

behaviour of all other users. For each $u \in Client$ a tuple of FACT models is synthesized as:

$$\mathcal{F}^m\big((T^u_{test}, T^u_{ref}), M_T\big) = (M^u_{test}, M^u_{ref}) \tag{7.10}$$

where M^u_{test} is a FACT model which characterises the behaviour of user u based on transitions observed in the partitioned log models. M^u_{ref} is then a model against which the behaviour of u will be compared[2]*.*

In order to analyse models using the FACT framework the model templates are stored as parametric FACT models. The parameters of these models are then the number of observed transitions in the sub logs of the trace partition. The model template may be generated from a UML activity diagrams such as that shown in our motivating example, Figure 7.1, and can be derived as follows.

First, we build the finite state set S comprising a state for each activity node from the diagram, and an initial state s_0 and a final state s_{F} for the initial and final nodes of the activity diagram, respectively; let $node(s)$ denote the activity diagram node associated with state $s \in S$. We then assemble the transition probability matrix $\boldsymbol{P}$:

- We set $\boldsymbol{P}(s, s') = 1.0$ for every pair of states $s, s' \in S$ for which the node reached immediately after $node(s)$ (i.e. without going through intermediate nodes associated with states from S) is *always* $node(s')$, and for $s = s' = s_{\mathsf{F}}$.
- We associate an unknown transition probability $\boldsymbol{P}(s, s')$ with each pair of states s, s' for which $node(s')$ can be reached from $node(s)$ going only through decision nodes in the activity diagram.
- We set $\boldsymbol{P}(s, s') = 0$ for every other pair of states $s, s' \in S$.

Next, we define a labelling function L that labels each state $s \in S$ with an atomic proposition that suggestively reflects the state of the system after the execution of the activity associated with $node(s)$. For instance, we used the label *Allocated* for the DTMC state associated with the activity AllocateToSupport of our Ticket Support process. Finally, we augment the resulting DTMC with cost/reward structures with reference to the policies to be analysed such as those found in Table 7.1.

Example 7.3. *Applying the DTMC derivation method described above to the Ticket Support activity diagram from Figure 7.1 yields the DTMC shown in Figure 7.4*[3]*. The 17 states of this DTMC correspond to the 15 activities of the business process plus the initial and final states of the activity diagram, and the labelling captures the state of a ticket during the execution of the process. The number combinations $r_1|r_2|r_3|r_4$ annotating the DTMC states define four cost/reward structures used to formalise the policies from Table 7.1: an "expensive" client structure (r_1, cf. policy P6 from Table 7.1), a "suspended" structure (r_2, cf. policy P2), a "reopened" structure (r_3, cf. policy P1), and a "lazy" support staff structure*

[2] We assume that the number of users acting abnormally is a small fraction of the total population such that any aggregated performance measure based on M^u_{ref} will not be materially affected by abnormal user activity when the population as a whole is evaluated.

[3] Note that the level of abstraction used here is different to that used when considering the continuous time properties of the IT support process where several states are combined to simplify the model without losing accuracy with respect to the properties of interest.

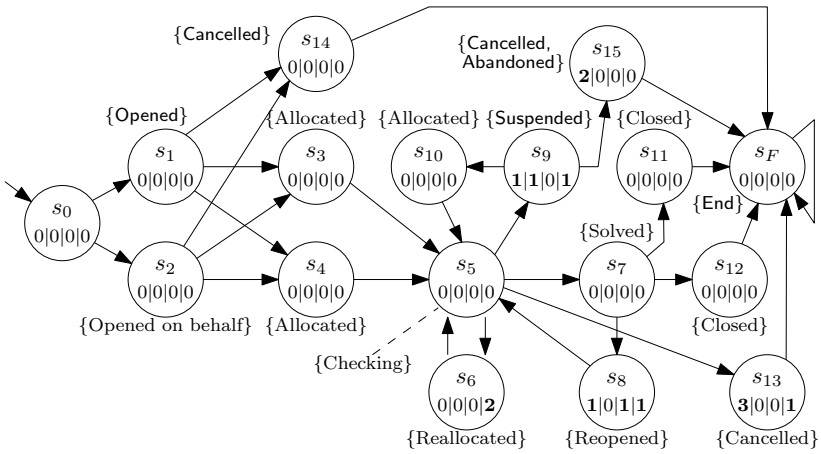


Figure 7.4: Parametric FACT model of the IT support system. Transition probabilities are parameters of our model (for states with multiple outgoing transitions) or 1.0 (for states with a single outgoing transition).

(r_4, cf. policy P5). For example, state s_6 (which corresponds to a ticket being reallocated by a member of the support team) is annotated with $r_1|r_2|r_3|r_4 = 0|0|0|2$, indicating that ticket reallocation is not characteristic of an expensive client ($r_1 = 0$), is not a reopen or suspension ($r_2 = r_3 = 0$), but is a strong characteristic of "lazy" support staff ($r_4 = 2$). Selecting values for rewards in the model requires domain expertise or a level of experimentation in order to obtain values which reflect the intention of the policy.

7.2.5 Step 3 : Model analysis

The next step in the process is to analyse the models generated in the previous step using an analysis function, $\mathcal{F}^a$. The function is configured through analysis parameters, $\mathcal{P}^a = (p_1, p_2, \cdots)$ where the number and form of parameters will vary depending on the type of analysis to be conducted. The analysis function $\mathcal{F}^a$ then generates a tuple of results as:

$$\mathcal{F}^a((M_1, M_2, \cdots, M_r), \mathcal{P}^a) = (R_1, R_2, \cdots, R_r) \tag{7.11}$$

where the type of each result object is defined by the analysis undertaken and the result quantifies the observed user behaviour.

Example 7.4. *In order to analyse policy P1 using FACT we need to provide two analysis parameters. A property specified in PCTL and a confidence level α. P1 may be encoded as a PCTL property,*

$$\Phi = \mathcal{R}^{\text{reopened}}_{=?}[\text{F End}] \tag{7.12}$$

and hence

$$\mathcal{P}^a = (\Phi, \alpha) \tag{7.13}$$

The FACT model checking engine will return one result for each model passed to it such that:

$$\mathcal{F}^a\big((M^u_{test}, M^u_{ref}), \mathcal{P}^a\big) = (R^u_{test}, R^u_{ref}). \tag{7.14}$$

where R^u_{test} and R^u_{ref} are confidence intervals for the property Φ at the given confidence level for the test and reference models, associated with a user $u \in Users$, respectively.

7.2.6 Step 4 : Comparer

The comparer function $\mathcal{F}^c$ accepts the results from the previous stage and a set of comparison parameters $CP = (cp_1, cp_2, \cdots)$ where the number and form of the parameters vary depending on the form of the results produced in the previous step and the type of comparison to be carried out.

The function then generates a set of users $U^T \subseteq Users$, associated with each log partition $T = (T_{label_1}, T_{label_2}, \ldots, T_{label_r})$, for which the comparison holds.

$$\mathcal{F}^c((R_1, R_2, \cdots, R_r), CP) = U^T \tag{7.15}$$

Example 7.5. *For policy P1, evaluated using the FACT model analysis engine, the results obtained are confidence intervals. In order to compare intervals we use the rules defined in (6.6). We therefore wish to establish if*

$$R_{test} > R_{ref} \tag{7.16}$$

and, therefore, the comparison parameter is:

$$CP = (>) \tag{7.17}$$

and the comparer function is:

$$\mathcal{F}^c((R^u_{test}, R^u_{ref}), (>)) = \begin{cases} u & \text{if } R^u_{test} > R^u_{ref} \\ \emptyset & otherwise \end{cases} \tag{7.18}$$

7.2.7 Step 5 : Executor

Where the comparer evaluates to true the users associated with the log partition that generated the results are stored in a list U^T. These users are then assumed to be behaving abnormally. For all $u \in U^T$ we wish to amend the permissions which they possess in order to protect the process. This is achieved through an executor which can amend their permissions through the application of a permission change rule, *CR*, specified in the policy using first order logic.

Example 7.6. *For policy P1 any users identified as abnormal, $u \in U^T$ with have the ability to open tickets on behalf of clients, $\{\mathsf{OpenOnBehalf}\}$, removed and hence the change rule CR is:*

$$\forall u \in U^T \bullet perm'(u) = perm(u) \backslash \{\mathsf{OpenOnBehalf}\}. \tag{7.19}$$

7.2.8 Formalised policies

In order to implement the saRBAC approach we must first define a model synthesis function $\mathcal{F}^m$, a model analysis function $\mathcal{F}^a$ and a comparer function $\mathcal{F}^c$. When using the FACT analysis engine we use the model synthesis method outlined in section 7.2.4. The analysis function is then implemented as an instantiation of the FACT analysis engine. Finally the comparer function utilises the rules from rules established in (6.6).

Table B.1 then shows the formal policies as derived from the informal policies given in Table 7.1. We note that the confidence level for each policy is unknown at this stage and must be obtained by calibrating the policies so as to minimise false positives and false negatives. We will consider the calibration of these values in the next chapter where we will evaluate the approach using real-world data.

We also note that Policies **P2**, **P5** and **P6** are multi-part policies in which the results of stage one of the policy influence stage two. When reading these policies we note that U^T in a filter refers to the users identified by the previous policy stage whilst U^T in the change rule refers the the users identified by the comparer in the current stage.

We also note that where multi-part policies are used the confidence interval in each stage can be different. Indeed Policy P5 indicates that a higher level of confidence is required to apply the change rule in part two of the policy.

7.3 Related work

The are several works on increasing the flexibility of access control mechanisms. These have been focused on adding dynamism to access control decisions, usually by means of incorporating risk and benefit values in the access control decision making process [144]. For example, Shaikh et al. [145] presented a method for risk-based access control decision, in which risk and trust are calculated based on historic user behaviour of granted access. Cheng et al. [146] employed Fuzzy Logic, which is quantified and used to define several thresholds according with risk tolerance. Based on trade-off analysis of risk versus benefit the solution grants access but with additional actions to mitigate risk depending on the threshold. Examples of such additional actions include: stronger logging, extra charge for the user, different access levels token. Kandala et al. [147] presents an abstract formal model for expressing risk-based access control policies. The model is then applied to UCON access control model [148], which is extended to accommodate risk-awareness.

However, there is a distinction between dynamic access control decisions and dynamic modification of access control policies. The approaches based on the first consider that risk related information are encoded in the access control policy beforehand, restricting the decision making process to whether or not to grant access to a particular request. On the other hand, it is possible to notice a number of works moving towards the dynamic modification of the access control policies in response to detected situations.

Bijon et al. [149] presents a formalisation of an adaptive quantified risk-aware RBAC system, identifying how to utilize estimated risk values and thresholds in the access control

decision making. Additionally, their approach also considers that risk values/threshold can be dynamically modified, identifying the need for monitoring, anomaly detection and risk re-estimation functions together with mechanisms to automatically revoke roles and permissions from users and roles respectively. Another approach is the Self-Adaptive Authorization Framework (SAAF) [141], which focus on adapting authorisation policies during run-time. SAAF has been demonstrated in the context of PERMIS authorisation system [150], in which access control policies are dynamically modified for dealing with insider threats, such as, an elevated number of downloads in a short time window. Ariadne [151],[152],[153] is an approach for dealing with security threats in Cyber-Physical systems by modelling the topology of the system (its physical objects and agents) together with its security requirements. This model is then used for conducting threat analysis and response planning at run-time about possible future changes in the topology, such as access of an agent to a particular area, for detecting violations of security requirements. Based on this, the approach suggests changes in the access controls rules for mitigating the identified threats.

Similar to those approaches, our work looks for means of adapting access control policies. However, different from them, we have focused on how to identify anomalous behaviour as triggers for adaptation. In this context, Legg et al. [135],[154],[155] presented an insider threat detection and analysis system. Their solution builds tree profiles of users based on different types of logging information, and applies a semi-supervised approach to assess how the current observations deviate from previously observed activities. In case of abnormality detection, the system raises an alarm. Probabilistic techniques have also been used for responding to network attacks, particularly as part of Intrusion Response Systems. [156] employs a Markov Decision Process as planning for deciding how to respond to a detected intrusion. Unlike this approach, our work focuses on insider threats operating inside trusted network boundaries, which the work in [150] does not consider.

Different from these approaches, we are looking at the application level, applying our approach to socio-technical systems based on business processes, where there is a strong human interaction. We also employ confidence level calculations for improving our decision making process. As we have more observations of normal behaviour, the confidence in our model ("profile") of normal behaviour and in making decisions based on it increases, allowing us to take different actions. Thus, adaptation is needed/useful. One clear observation from these related work is that access to real-world data is difficult, and thus, researchers synthesize data that are similar to that of a real-world enterprise, or use a subset of data points, or apply insider threat detection techniques to other problem domains. In our work, we gather a variety and volume of data observed in a modern real-world organization.

Chapter 8

Evaluation of the saRBAC Approach

We evaluated saRBAC with a prototype implementation within the information system running the IT support business process at the Federal Institute of Education, Science, and Technology of Rio Grande do Norte (IFRN), Brazil. Because of the business-critical nature of the system, we ran saRBAC as an advisory system suggesting access control modifications that IT managers could verify instead of implementing them directly.

8.1 Implementation

Our collaborators at IFRN implemented a prototype for saRBAC using Python and followed a service-oriented architecture as shown in Figure 8.1. This architecture allows for the deployment of saRBAC components across several processing nodes, and thus the parallel analysis of multiple users. The prototype was integrated into the unified system for public administration (*SUAP*) as an advisory tool for IFRN management.

The *Manager* component co-ordinates activities in the saRBAC approach and interfaces with the system under analysis, represented by the *SUAP*. The *Manager* also takes as input the

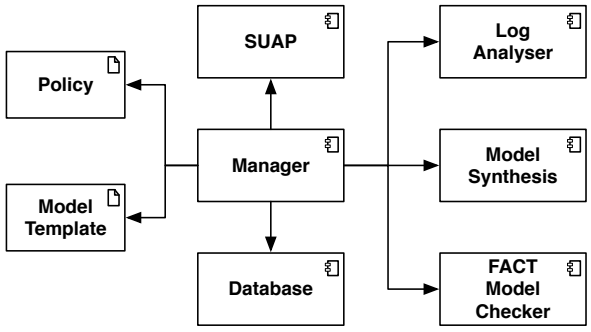


Figure 8.1: saRBAC architecture

policy for analysis and a model template. The *Database* component stores the intermediate data produced during the analysis process.

The *Manager* accesses logging information from *SUAP* which contains the execution traces of all business processes undertaken. The *Manager* passes the filter definition to the *Log Analyser* component in order to partition and filter log data for users identified in the policy definition. Trace sub sets are passed to the *Model Synthesis* component along with the model template in order to construct FACT models for analysis. These models are analysed by the *FACT Model Checker* and the resultant confidence intervals are returned to the *Manager* which implements the comparison function and thus generates an RBAC adaptation recommendation.

When a policy contains more than one rule, the *Manager* processes one rule at a time parsing and chaining rule stages as appropriate.

8.2 Evaluation

To evaluate saRBAC, and demonstrate its feasibility to the IT management directorate, a series of experiments using the prototype implementation were conducted by our collaborators at IFRN using real data from SUAP. This section summarises the main results obtained in these experiments, divided in three parts. We begin by describing the experiments conducted to calibrate the confidence levels used for each policy (Section 8.2.1), followed by results about the use of the approach after calibration (Section 8.2.2). Finally, we present experiments that assess the performance of saRBAC (Section 8.2.3).

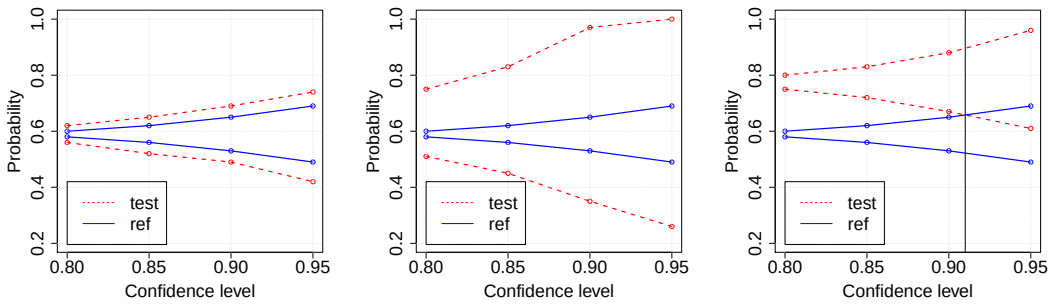
8.2.1 Preliminary experiments and confidence level calibration

For each policy it is necessary to calibrate the confidence level used to identify deviations in user behaviour whilst minimising the number of false positives.

Let us consider a policy which examines if the probability of a user re-opening a ticket is higher than normal. The complete log is partitioned into two trace sets for each user such that a partition for a user u is (T^u_{test}, T^u_{ref}). Models (M^u_{test}, M^u_{ref}) are then built for each trace partition and analysed with reference to a PCTL property using FACT. Since we do not know the confidence level to select we generate intervals at four confidence levels, $\alpha = \{0.8, 0.85, 0.9, 0.95\}$. For the example policy we are interested in triggering the policy only if $(R_{test} > R_{ref})$. Since R_{test} and R_{ref} are intervals we use the interval comparison function (6.6). Figure 8.2 then shows the confidence intervals at each confidence level for three simulated users.

From Figure 8.2 we can see that, in all cases, as the confidence level is increased the intervals associated with the property analysis widen. We also note that the intervals associated with the reference trace set are typically narrower than those associated with the test trace set. This is because the reference set contains a larger number of observations.

The user shown in Figure 8.2a represents a user with normal behaviour. The results confirm



(a) (b) (c)

Figure 8.2: Comparison of confidence intervals for a set simulated users where (a) is a user behaving normally , (b) is a using behaving abnormally for whom insufficient observations exist to identify abnormal and (c) is a user behaving abnormally and behaviour.

this since the intervals overlap across the range of confidence levels considered. This means that there is no evidence that the results for this user are abnormal. We would not want the policy to be triggered at any confidence level for this user and, based on this data. the policy is never triggered.

The user shown in Figure 8.2b represents a user with abnormal behaviour for whom little observation data is available. This lack of observation data means that analysis of the test model returns wide intervals. These intervals then overlap the results for the reference model across the range of confidence levels considered. If we were to reduce the confidence level significantly, e.g. to 0.3, then we may find a level at which it was possible to classify the user as acting abnormally. However at this lower level we are also likely to identify a large number of false positives. While failing to identify this user as abnormal may be considered as a false negative this feature of saRBAC can be seen as a positive advantage. When policy adaptations are triggered they typically restricted the activities a user can undertake, or add additional checks to the existing process. This is likely, therefore, to increase the work load for the team as a whole and make the process less efficient. In addition the policy restriction may trigger user reviews which are costly to the organisation. This level of impact is unlikely to be acceptable based on limited observation data.

Finally the user shown in Figure 8.2c represents a user who displays abnormal behaviours and for whom a reasonable amount of observation data is available. Using the interval comparison function (6.6), we can see that ($R_{test} > R_{ref}$) returns true for all confidence intervals up to 0.90. At higher levels the intervals overlap and there is insufficient evidence to say that the policy is violated [157].

Since we know that the policy should be triggered for this particular user we can use this as the basis for calibrating the confidence level in the policy. Selecting a confidence level α implies that there is evidence at all levels below α for the policy to hold. A confidence level of 0.9 would then cause the policy to trigger for this user and for all users for whom there is at least this level evidence of abnormality. Choosing a lower level may result in false positives whilst selecting a level greater than 0.9 would not trigger the policy for this user.

Using a single user for calibration is not advisable however, since the user selected may not be representative of the population at large. Where many observations exist for a users displaying abnormal behaviour the intervals may never overlap, even at very high confidence levels. Using such a user for calibration may result in a large number of false negatives. Indeed, it may not be possible to select a confidence level that produces zero false negatives and false positives. Selecting a suitable confidence level for a policy may therefore require a level of subjective analysis.

The automatic calibration of confidence levels remains a topic for further work and therefore in our evaluation a manual calibration was carried out as follows.

Data collected for 63 users of the system was considered over a time window of 30 days and, for users with the *Client* role, three different subsets were considered. Initially, we selected users that were known to the management team for displaying undesirable behaviours. We refer to these as "famous" users in the rest of the section. A second subset was defined as the most active users in which we selected the users with the largest volume of handled tickets. Finally, we selected two random clients, not present in the other sets.

For the *Support* role, a subset of famous users was also identified as well as the most active users. We then chose five users who had handled a similar volume of tickets to those famous users. Finally two random support users were selected.

In addition we created several synthetic users to exercise specific policies deemed important by the IT managers, but which were not "triggered" by any current support user, e.g. policy P3.

Tables 8.1 and 8.2 present the saRBAC analysis results for these users in the *Client* and *Support* roles, respectively. The ID for each user is anonymised in both tables. For each user and policy, we computed confidence intervals for the examined behaviour characteristic at multiple confidence levels, and the tables report *the highest confidence level, α, for which the policy is triggered*, i.e. the confidence level at which the interval result obtained for the test traces is outside the interval obtained for the reference traces.

A '0' indicates that the user does not trigger the policy at any confidence level between 0.75 and 0.99. To decide the confidence level threshold of each policy, we started to compute confidence intervals at a confidence level $\alpha = 0.99$ and analysed the obtained results, identifying any false positive/negatives.[1] Next, we lowered the confidence level to 0.95 and then in steps of 0.05 repeating the analysis at each stage.

We start with the policies targetting the *Client* role, whose results are shown in Table 8.1. For policy P1 we wish to identify clients who reopen tickets more frequently than normal. A manual analysis of the trace logs was conducted for the famous users as a baseline for calibration. This resulted in an expectation that, for policy P1, we would expect users 101701, 101702 and 101704 to trigger the policy and that user 101703 would not. The results for users 101702 and 101704 are as expected, with both users triggering the policy (at 0.95 and 0.9 confidence levels, respectively). The result for user 101703 was also in line with expectations and this user did not trigger the policy at any confidence level considered.

[1] This involved an independent manual analysis of the activity of these users in consultation with the IT management team. This discussion was undertaken by our collaborators at IFRN.

Table 8.1: Results obtained for the policies targeting the *Client* role. Highlighted cells indicate those users that we wish to trigger the policy.

User type	User ID	P1	P6
famous	101701	0.8	0.95
	101702	0.95	0.95
	101703	0	0
	101704	0.9	0
most active	101705	0.85	0.95
	101706	0.95	0.95
	101707	0.8	0.95
	101708	0	0
	101709	0	0
random	101710	0.95	0
	101711	0	0.8

Whilst user 101701 did trigger the policy, we found that the level at which the policy was triggered was lower than that found for others users, i.e. 0.8 rather than 0.9. We therefore repeated our manual analysis of the trace logs for this user. This confirmed that the result produced by our approach was correct. User 101701 should *NOT* trigger policy P1. This discrepancy was due to an error in the initial manual analysis process which is rather time consuming and complex. Indeed this illustrates one of the benefits of the formal analysis undertaken by our approach.

From the most active users, only user 101706 triggered policy P1. Whilst this was an unexpected result, a manual inspection of the user's activity traces confirmed this as correct. The same was found for user 101710 from the random subset. Users 101705 and 101707 trigger the policy if the confidence level associated with the policy is reduced. Discussions with the IT managers and manual analysis of the trace logs for these users agreed that the policy should not mark them as abnormal and to do so would be deemed a false positive. A confidence level of $\alpha = 0.9$ was therefore selected for policy P1.

Policy P6 was designed to detect clients whose behaviour is considered expensive within the Ticket Support System. This expense is encoded through rewards in our FACT model which are associated with reopening, suspending and cancelling tickets. The results from Table 8.1 are in line with a manual analysis for the famous user set with users 101701 and 101702 triggering the policy at a confidence level of 0.95.

For the most active users, the results were unexpected, but confirmed by manually inspecting the behaviour of the "offending" users (i.e. 101705, 101706, 101707). For the random users we note that user 101711 would trigger the policy if the confidence level were 0.8, a false positive, and therefore a confidence level of $\alpha = 0.9$ for policy P6 was selected.

The results for the policies targeting the *Support* role are presented in Table 8.2.

Policy P2 handles support attendants with high rates of suspended tickets. Manual analysis of the data highlighted users 201703 and 201708 had a suspension rate of over 50% and these users should therefore trigger the policy. After an analysis of the results obtained, a confidence level of 0.8 was selected, which correctly identified these as "offending" users and noted that lowering the confidence level to 0.75 would produce three false positives for the

Table 8.2: Results obtained for the policies targeting the *Support* role. Highlighted cells indicate those users that we wish to trigger the policy.

User type	User ID	P2	P3	P4	P5	
famous	201701	0	0	0	0.95	0.95
	201702	0.75	0	0.8	0.99	0.99
	201703	0.85	0	0	0.99	0.99
	201704	0.75	0	0.8	0.99	0.99
	201705	0.75	0	0	0.99	0.99
most active	201706	0	0	0.85	0.99	0.99
	201707	0	0	0.85	0	0
	201708	0.8	0	0	0.99	0.99
similar	201711	0	0	0	0.90	0.90
	201712	0	0	0	0	0
	201713	0	0	0	0.99	0.99
	201714	0	0	0	0	0
	201715	0	0	0	0.85	0
	201716	0	0	0	0.99	0.99
synthetic	x1	0	0.95	0	0	0
	x2	0	0.95	0	0	0
	x3	0	0.9	0	0	0
	x4	0	0.85	0	0	0
random	201721	0	0	0	0.99	0.99
	201722	0	0	0	0	0

dataset considered.

Policy P3 deals with *Support* users that cancel tickets without requiring more information. This is a very unusual behaviour for the Ticket Support System, however, guarding against it was explicitly requested by the management team. Since this policy was not triggered by any of the users,several synthetic users were created to ensure that the policy would trigger as expected. Four users were synthesized: two offenders, and two regular users with similar levels of activity a confidence level of 0.95 was then selected to separate the two types of user. Since the results for this policy were obtained with synthetic users, it will require careful monitoring and revision. Examining a larger dataset, i.e. the complete user base of SUAP, may allow us to identify users displaying this behaviour.

Policy P4 deals with support users that open tickets on behalf of clients where the ticket is subsequently abandoned. This is considered suspicious and we were not aware of any users displaying this behaviour before running the experiment. However, the approach identified four users with this behaviour. Users 201702 and 201704 trigger the policy at a confidence level 0.8. A manual analysis of these users and discussion with the management team, however, decided that these would be a false positive. Users 201706 and 201707 require a higher confidence level to trigger the policy (0.85) and these are indeed displaying abnormal behaviour. Selecting a confidence level of 0.85 is therefore able to distinguish between real and false triggers for this dataset.

Finally, policy P5 addresses support users that are "lazy", which according to the management team is a support user with a high combined rate of suspended, cancelled, reopened and reallocation tickets (cf. the "lazy" cost/reward structure from our DTMC model Figure 7.4). An analysis of the results obtained showed that a confidence level of 0.95 in stage

Table 8.3: Confidence levels for each policy after calibration.

Policy	Confidence Level
P1	0.90
P2	0.80
P3	0.95
P4	0.85
P5.1	0.95
P5.2	0.99
P6	0.90

one of the policy and 0.99 in stage two resulted in the policy being triggered as expected for the calibration set. Again we found users which had not been expected to trigger the policy, but the policy was proved correct when validated through manual data analysis. We note however that the level of confidence is very high and may thus lead to false negatives. A larger dataset may be required to fully calibrate this policy.

We wanted our approach to be able to differentiate between users whose behaviour is within acceptable limits of normal and those whose behaviour is abnormal and likely to pose a risk to the operational process. We also required that the policies were resistant to false alarms. We believe that the calibration process has achieved this. The experiments presented so far calibrated the confidence level threshold for each policy from Table B.1 with the resultant confidence levels given in Table 8.3. Using these calibrated confidence levels we were able to distinguish between users who are acting normally with respect to the reference set and those who exhibit abnormal behaviour.

8.2.2 Validating the confidence interval

Having identified which users act normally and abnormally with respect to a policy during the calibration stage we now wish to show that saRBAC is applicable to other trace sets. Using a second dataset gathered for these users then allows us to validate that the policies are not over-fitted to the calibration dataset.

The calibration dataset consists of half the data gathered from the IT support process, days 1–30. We can therefore use the remaining data to validate saRBAC. In this section we consider two new datasets for testing. The first consists of those log entries generated in days 31–45. By selecting a dataset which considers a smaller time window than the calibration set we are able to identify if 30 days is required to identify abnormal behaviour. Using a smaller set will widen the confidence interval associated with model analysis but, if abnormality can be detected in this smaller set, then the speed with which abnormal behaviour could be detected will also be increased.

The second dataset consists of traces from days 31–60. This second set which, like the calibration set, consists of 30 days allows us to consider if the results found in the previous section are applicable to other datasets from the same system captured at other points in time.

Having identified normal and abnormal behaviours for users during the calibration stage we

Table 8.4: Policies for the support role where #1=policy triggered on by test set one and #2=policy triggered on test set two

User type	User ID	P2	P4	P5	
		α=0.8	α=0.85	α=0.95	α=0.99
famous	201701	–	–	–	–
	201702	–	–	#2	#2
	201703	#2	–	#1, #2	#1,#2
	201704	–	–	–	–
	201705	–	–	–	–
most active	201706	#2	#1, #2	#1, #2	#1, #2
	201707	–	#1, #2	–	–
	201708	#2	–	#1, #2	#1,#2
similar	201711	–	–	–	–
	201712	–	–	–	–
	201713	#2	–	#1, #2	#1,#2
	201714	–	–	–	–
	201715	#1	–	#1	–
random	201721	–	–	#1, #2	#1,#2
	201722	#1	–	–	–

would expect the behaviour of these users to continue in the period over which data for the validation set was gathered. In the following analysis we therefore highlight those users where this is not the case. Where a user is identified as no longer triggering a policy, or where a user triggers a policy who did not previously do so, we manually analyse the trace set and, in discussion with the management team, decide if saRBAC is operating as intended.

Whilst it would be beneficial to increase the size of the validation set to include additional users not included in the calibration set, the manual analysis and subsequent discussion stage is resource intensive. Due to the limited resources available this was not possible and remains an area for further evaluation.

In this section we consider only those policies which concern support users and omit policy P3 as this only considers synthetic users. We also note that user 201716 is omitted from the validation set since the user was away from the office in the period over which this data was gathered.

Table 8.4 shows the support users that triggered each policy for the analysis carried out using each of the validation datasets.

Policy P2, ticket suspensions, is triggered for users 201703 and 201708. These are the same users identified using the calibration set. We note however that the abnormality was not triggered for the 15 day set (set one) indicating that, for these users, their is insufficient evident evidence gathered in the shorter time window. This is not surprising since a lower number of observations leads to a wider confidence interval and therefore the likelihood of overlapping with the normal behaviour set is increased.

For the other four users identified (201706, 201713, 201715 and 201722) we find that the analysis is working as expected and that in these time periods the behaviour identified is different to the "normal" behaviour. Given that the process is undertaken by human operators it is not surprising that the behaviour is inconsistent across time periods. It is interesting to note that for users 201715 and 201722 the abnormal behaviour was not

Table 8.5: Results obtained for the performance experiments (reported in seconds, as mean time ± standard deviation)

Policy	Log analyser	Model Synthesis	FACT	Total
P1	0.00069 ±0.00017	32.71 ±0.53	30.65 ±0.20	63.37 ±0.68
P2	0.00059 ±0.000010	32.46 ±0.31	30.80 ±0.21	63.27 ±0.36
P3	0.00048 ±0.0000087	32.45 ±0.35	30.61 ±0.1127	63.06 ±0.3433
P4	0.00056 ±0.00004	32.74 ±0.50	30.69 ±0.31	63.43 ±0.81
P5	0.00060 ±0.000024	32.52 ±0.2201	30.71 ±0.04907	63.24 ±0.2875
P6	0.0005 ±0.000007	31.86 ±0.84	30.76 ±0.20	62.63 ±0.73

noticed when considering the longer time window. This would suggest that the size of the time window is a factor in determining the behaviour of individuals. Further work is required to identify the optimal time window to use and this may be on a per policy basis. Using the formal specification defined in the previous chapter these time windows can be added to policies as log filters.

For Policy P4, abandoning client tickets, we find that the users identified are consistent with the calibration set and users are identified using both the 30 and 15 day time windows.

For Policy P5 we find that a number of users are consistently identified across both calibration and validation tests, 201702, 201703, 201706, 201708, 201713 and 201721 and that for user 201715 the first part of the policy is triggered (for the short time window) which is similar to the calibration set, i.e. for calibration there is some evidence of abnormality in this stage but insufficient to trigger the policy. We also not that for user 201702 there is only sufficient evidence of abnormality over the longer time window.

Users 201704 and 201705 do not trigger the policy in this stage. This could be because of a change in user behaviour for the time periods considered, or because the confidence level, $\alpha = 0.99$ requires further calibration using a larger dataset.

8.2.3 Evaluation of saRBAC performance

We also conducted experiments to assess the computational overheads of our approach. The experiments examined the performance of the saRBAC prototype when running on a single processing node, by measuring the execution times of its three main components, i.e. Log Analyser, Model Synthesis and FACT Model Checker. All experiments used real log data from SUAP over a 30-day time window (the same window considered for the calibration experiments), and involved a total of 919 tickets and 63 users (23 support users and 40 client users). All experiments were executed on 2.2 GHz Core i7 computer with 8GB of RAM, and were repeated three times.

Table 8.5 presents the mean and standard deviation of the component execution times in seconds. The analysis of each policy takes approximately one second per user (above 60 seconds for the 63 users). While this overhead may seem too high for large business processes, it must be noted that saRBAC is meant to run infrequently (e.g. every few days) because the detection of insider threats within business processes with infrequent user activities requires multiple days of logs. Moreover, the performance of saRBAC can be improved dramatically by carrying out the essentially independent analyses required for different policies and different users concurrently, potentially after refactoring the architecture of our saRBAC prototype in line with existing self-adaptive system architectures [158],[159],[160],[161].

8.3 Threats to validity

8.3.1 External validity

External validity threats may arise for our approach if the characteristics of the system under test are not representative of a wider range of systems.

By applying our approach to data logs obtained from a real-world system we have shown that the approach is not limited to theoretical processes with well behaved parametric distributions. We also used a range of policies for the system analysis to demonstrate that the formalisation defined in Chapter 7 is sufficient to capture real-world concerns, including policies which require multiple stages of analysis. Although we believe that the approach is applicable to a wide class of system processes, SUAP may not be typical of other business critical processes. Further evaluation is required to apply saRBAC to other system processes in order to evaluate the generalisability of our approach.

Another external threat may arise if models required to analyse the system under review are too large for parametric model checking. We believe that under such cases alternative, simulation based, model analysis techniques may be substituted for FACT and the modular nature of our approach and implementation would allow for such techniques. This remains, however, an area for further work.

8.3.2 Construct validity

Construct validity threats may arise due to the assumptions made when collecting data for our experiments or when defining the policies for analysis. To address these problems we made use of real-world data logs extracted from the SUAP system to evaluate our approach. The policies were then developed in consultation with the management team at IFRN. Certain policies were rejected from the implementation as they were not suitable for analysis using DTMC approaches, e.g. simple log counts, however by choosing six policies we where able to examine two different groups of users, clients and support, and a range of property types for analysis. In addition the dataset obtained was split into two sets for confidence level calibration and test/validation.

8.3.3 Internal validity

Threats to internal validity may arise from our bias or in our interpretation of the experimental results. The use of confidence intervals to classify difference in observed behaviours is based by formal proofs and FACT itself is a formal verification technique which shows the effects of observation uncertainty on system performance.

Our selection of confidence levels for policies is a manual process at present and although a heuristic has been used to select the level, this requires further work to ensure the process is not only robust but free of bias since it involves manual analysis and a subjective interpretation of data.

Part IV

Conclusions and Further Work

Chapter 9

Conclusions

In this thesis, we introduced a range of tool-supported techniques for the modelling and analysis of real-world systems. These techniques integrate logs of observed behaviour with tried and tested probabilistic model checking to provide models and analysis results which more accurately reflect real-world behaviours. We demonstrated how using observation data in both CTMC and DTMC models can support improved decision making by avoiding the incorrect decisions which arise from using the analysis of models which do not fully capture the complexities of real-world systems and processes.

For continuous time systems we showed how transient quality-of-service properties may be more accurately assessed by constructing phase-type distribution models for process activities. This technique is applicable to processes where the structure of the process is well known but the temporal characteristics are poorly represented by traditional modelling techniques. The phase-type distribution models are fitted to logs of timing data captured from real process instantiations, ensuring that the models generated reflect the behaviour observed.

For discrete time systems, we demonstrated how observation data can be combined with formal verification techniques to allow for formal verification with confidence intervals. We then showed how this technique could be used to support the identification of abnormal behaviour patterns in a real-world IT support processes. By developing a formal definition for abnormality detection policies, we showed how our approach is able to capture real-world requirements, detect negative behaviours and limit the potential for users to compromise operational processes.

In the remainder of this chapter we summarise the main contributions of the thesis and the part they play in supporting observation-enhanced verification of operational processes.

We developed OMNI, an observation-based CTMC refinement method that significantly improved the accuracy with which transient quality-of-service properties can be analysed using CTMC models. The technique makes the CTMC more realistic through the addition of states and transitions in the model which more accuracy represent the temporal characteristics of the process activities. Adding states to the model not only increases model accuracy but also increases the computational costs required for model-analysis. Our approach therefore offers a trade off between model accuracy and size. OMNI uses a two stage approach for

refinement. The first stage addresses the problem of modelling areas of zero density in the behaviour of real-world activities. This is required since real activities rarely have non-zero minimum execution times i.e. deterministic delays. Erlang distributions are used for this purpose and we demonstrate how the error associated with the delay modelling may be computed. Selecting the appropriate refinement level for the delay modelling is then achieved through the selection of a single parameter which defines the number of states to add to the model. The second stage refines the CTMC further through the use of phase-type distributions which model the observed holding time for each activity. Phase-type distributions can approximate any continuous distribution, with a strictly positive density, arbitrarily closely. This may require an infinite number of states to be added however. We therefore developed a fitting algorithm which allows for bounds on the size of the model produced and uses a refinement parameter to specify the termination criteria as a bound on the fitting error.

Controlling the size of models produced by OMNI is further enhanced through the use of property-centric model refinement. This approach reduces the size of CTMC models produced by OMNI significantly and does so without impacting on the accuracy of the analysis provided. Our approach uses the CSL property, which encodes our transient quality-of-service requirement, and a high level CTMC model of the process to classify states in the model and therefore the activities which they represent. Where states are identified as having no impact on the property analysis their refinement is not required hence reducing the amount of computational effort required for analysis. Where states appear only once on all paths which satisfy the CSL property the need to model delays associated with those states is removed by amending the CSL property. Finally those states which we define as "together" sets may have their delays combined. Since modelling delays is computationally expensive this again reduces the size of the model and the computational expense of verification. Since the approach makes use of the high level CTMC for the classification process this requires little computational effort and the reduction in size of the resultant models for the systems considered is significant. We also showed how activity refinements can be cached such that multiple properties can be analysed without the need to refine activities more than once.

Whilst the property-centric refinement stage of OMNI helps to control the size of the models produced, this increase in OMNI model accuracy comes at a cost. The increased size and complexity of models leads to an increase in the amount of computational power required to analysis these refined models. For operational processes, however, the size of the high-level abstract models are typically small and, as such, the models produced by OMNI are still analysable with modest resources.

The techniques developed for OMNI were implemented in a tool which we have made freely available. In order to evaluate the tool we developed two case studies. The first utilises real-world web-services composed as a process for services requests on a web-application. The second was developed using a real-world IT support system implemented at a University in Brazil. The case studies are from different domains demonstrating the generalisability of the approach. Our experimental results showed that OMNI-refined models support the analysis of transient QoS properties of processes with greatly increased accuracy compared to the high-level CTMC models typically used. Furthermore, we showed that significant accuracy improvements are achieved even for small training datasets. Where computational resources are at a premium, we showed that using a coarse-granularity for refinements produced relatively modest increases in model size whilst still improving the accuracy of model-analysis.

For discrete time systems we developed a tool for the formal verification of discrete time Markov chains with confidence intervals. The tool, which we call FACT, builds on the theoretical framework developed by Calinescu et al. [23]. FACT accepts a parametric DTMC model defined in an extended version of the PRISM language which allows for the specification of transition observation counts. This avoids the need for point estimates for transition probabilities and enables the calculation of confidence intervals. We demonstrated the use of FACT using a case study concerning a business critical web application. The PRISM language is reused where possible to aid in model understanding and requires no modification to the PCTL formulae. We demonstrated how the confidence interval results can be interpreted and how invalid decisions may be avoided. Finally we showed how FACT can be applied in a range of contexts by applying FACT to five systems from multiple application domains. The tool and the models from each application domain have been made freely available.

Having shown how confidence intervals may be derived for models where observation data exists we then developed an approach for the identification of abnormal behaviours using the FACT tool. Our motivating example was based on a real-world IT support system and the requirements considered were generated in consultation with the IT support management team. Our approach involved the development of a formal policy framework which was able to encode a range of policies. These policies serve as the input to our approach along with the trace data obtained from the systems and a parametric DTMC derived from the UML description of the operational process. We showed how filtering and partitioning rules could be defined to allow for the analysis of user behaviours from the source trace set. We then demonstrated how rules for model analysis and result comparison could be encoded. Finally we showed how change rules could be defined within a role based access control system in order to limit user permissions and hence mitigate against invalid behaviours.

We have evaluated our approach using data from a real-world system and examined the behaviour of a set of users using the formal policies we defined. A prototype implementation of the system was built and deployed in an advisory manner on the IT support system. The policies defined required an initial manual calibration of the confidence level and we did this using a set of data extracted from the log specifically for this purpose. The policies were able to identify those users exhibiting harmful behaviours and indeed found a number of users who had not previously been identified. A second log set was then used to evaluate the approach and again users were correctly identified.

In order to implement the saRBAC approach a number of manual activities must be undertaken which are resource intensive i.e. modelling the process, instrumenting the system, defining formal properties from narrative specifications and calibrating parameters for saRBAC. This expensive will only be justified in processes for which abnormal behaviours present significant risks to operational performance.

Finally the web based travel application and IT support system case studies used to evaluate OMNI and saRBAC have been made available with all relevant log data. We envisage that these case studies will be used by other researchers to gain an insight into the development of modelling and verification techniques for real-world processes.

Chapter 10

Future Work

There are multiple ways in which the research presented in this thesis may be refined and extended. In this chapter we present those areas which we believe are the most significant research directions.

10.1 Updating observation-enhanced models at runtime

The techniques presented in this thesis assume that the characteristics of process activities remain relatively constant over time. In practice this assumption may not hold in all contexts due to: the dynamic nature of the environment in which the process is deployed; the experience of users who enact activities; or because the characteristics of mechanical and software components used in the process change over time. One important research direction, therefore, is found in the investigation of techniques which are able to update models so that they more accurately reflect the current state of dynamic real-world processes.

In practice the nature of the change to which the process activities are subject is unknown. Changes may be broadly characterised as a drift in the observed distribution, e.g. increasing load, or as point changes due to the process entering a new mode of operation, e.g. change of personnel or configuration. When activities are subject to drift, models may need to be generated continuously at runtime with the data upon which models are based extracted from observations using a moving window. A longer window provides more data for the model fitting algorithms but is slower to react to changes while a shorter window responds quickly but may provide insufficient data for accurate modelling. Windows may also be shaped to weight the value of data such that more recent observations are given a higher weight. For systems which move between distinct modes of operation change point detection can be difficult when the change is triggered by external factors. Once detected, all of the data previously observed may be invalid for predicting future performance. In such cases new models will need to be constructed using very limited data.

A number of such techniques have been previously developed for the runtime adaptation of models [122],[123],[124],[125],[105] and integrating these with OMNI and saRBAC may increase their applicability to a wider range of domains.

10.2 Additional application domains

In this work we have demonstrated that both OMNI and saRBAC can be applied to a real-world IT support system. We have also shown that OMNI is able to capture the temporal characteristics of software components and those of human centric processes. To further assess the generalisability of these approaches we would like to examine a number of additional domains and this presents another area for future research. For saRBAC one such domain may be health care. We would expect an increased level in the complexity of processes used in this domain when compared to the IT support system used in our study. The policy requirements for health care are also likely to be more complex and would help to validate our formal policy encoding.

10.3 Efficient parametric model checking techniques

Increasing the complexity of the domains for which the saRBAC approach is applicable will also require the development of new and novel parametric model checking (PMC) techniques. Whilst PMC techniques have seen significant advances in recent years [114],[162],[113] they are still computationally very expensive.

In recent work [5] we have shown how domain-specific modelling patterns may be exploited to provide efficient PMC and provide significant reductions in the time taken for the model checking of component based systems. Integrating these techniques with FACT and saRBAC is therefore an interesting area of future work and may be essential in order to allow for more complex problems to be analysed.

10.4 Alternative model analysis techniques in saRBAC

The saRBAC framework presented in this thesis has been instantiated using FACT as the model analysis engine of choice and, as such, the requirements encoded are limited to those which may be represented using discrete time Markov chains. However, the framework allows for the model synthesis and analysis stages to be replaced by alternate modelling formalisms. Using CTMCs rather than DTMCs, for example, would enable us to reason about the normality of the temporal characteristics of users within the process, e.g. Is the time taken for a client user to mark a ticket as complete higher than normal? The framework is not limited to Markovian models and we could also apply traditional statistical tests such as outlier detection or the Mann-Whitney test to allow for a wider range of policies to be encoded.

10.5 Verification with confidence intervals for CTMCs

The FACT model checker has been shown to be applicable in a range of contexts and effective as the model analysis engine for the saRBAC approach. FACT is, however, only applicable

to DTMC models. Parameter uncertainty also exists within CTMC models affecting the quality of verification results. Extending FACT to CTMC models therefore represents an opportunity for further research. This in turn would allow OMNI to provide verification results which quantify the uncertainty associated with transition probabilities within the high level abstract model. FACT for CTMCs could then be used to identify abnormal behaviour in saRBAC for policies which consider temporal features of CTMC models.

At present probabilistic model checkers provide limited support for parametric model checking of CTMC models and, as such, new techniques will need to be developed to transfer FACT into the continuous domain.

10.6 Unstructured data sources

For both the OMNI and saRBAC approaches presented in this work we have used observation data extracted from structured data sources. Recent advances in machine learning have introduced techniques which allow for data to be extracted from unstructured sources such as Twitter [163],[164],[165]. An opportunity exists therefore to integrate these data sources with probabilistic model checking in order to derive model parameters from natural language sources. We have recently undertaken preliminary work in this area to produce a prototype for the continuous planning of operational processes. Our solution applied natural language processing to Twitter data to derive parameters for a route planning model represented by a DTMC. The data extracted from Twitter was used to derive probabilities and reward structures within the DTMC model. In this way events were detected in the Twitter stream and used to trigger model updates and a re-verification of the model. From the updated model we generated revised plans which better reflect the status of the real-world.

Appendix A

MLE estimation of rate parameter for an exponential distribution

A random variable X is exponentially distributed with rate parameter *lambda*.

The likelihood function for λ given an independent and identically distributed distributed sample $x = (x_1, x_2, \cdots x_n)$ drawn from the variable is:

$$L(\lambda) = \prod_{i=1}^{n} \lambda \mathsf{exp}(-\lambda x_i) = \lambda^n \mathsf{exp}\left(-\lambda \sum_{i=1}^{n} x_i\right) = \lambda^n \mathsf{exp}(-\lambda n \bar{x}) \tag{A.1}$$

where

$$\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i \tag{A.2}$$

is the sample mean.

The derivative of the likelihood function's logorithm is:

$$\frac{d}{d\lambda} \mathsf{ln}(L(\lambda)) = \frac{d}{d\lambda}(n \mathsf{ln}(\lambda) - \lambda n \bar{x}) = \frac{n}{\lambda} - n\bar{x} = \begin{cases} > 0, & \lambda < \frac{1}{x} \\ = 0, & \lambda = \frac{1}{x} \\ < 0, & \lambda > \frac{1}{x} \end{cases} \tag{A.3}$$

Hence the maximum likelihood estimate for the rate parameter is:

$$\lambda = \frac{1}{\bar{x}} = \frac{n}{\sum_{i=1}^{n} x_i} \tag{A.4}$$

Appendix B

Self-Adaptation policies

Table B.1: Formalised self-adaptation policies for the Ticket Support business process using the FACT analysis engine.

Policy		Value	Description
P1	*filter*	$\{(T_{test}, T_{ref}) \in \mathbb{P}\mathcal{L}\times\mathbb{P}\mathcal{L} \mid \exists u \in Client \bullet (T_{test} = \{(id, \langle e_1, e_2, \dots e_N\rangle) \in \mathcal{L} \mid \exists 1 \leq j \leq N \bullet e_j.user = u\} \wedge T_{ref} = \mathcal{L} \setminus T_{test})\}$	Partition the complete trace set to compare a single client user with other users;
	$\mathcal{P}^a$	$(\mathcal{R}^{\text{reopened}}_{=?}[\text{F End}], \alpha)$	Evaluate the "reopened" reward with a confidence level; $(1-\alpha)$
	CP	$(>)$	Compare the results to see if client reopens tickets more often than normal;
	CR	$\forall u \in U^T \bullet perm'(u) = perm(u) \setminus \{\text{Open}\}$	If so then remove the ability to open new tickets.
P2.1	*filter*	$\{(T_{test}, T_{ref}) \in \mathbb{P}\mathcal{L}\times\mathbb{P}\mathcal{L} \mid \exists u \in Support \bullet (T_{test} = \{(id, \langle e_1, e_2, \dots e_N\rangle) \in \mathcal{L} \mid \exists 1 \leq j \leq N \bullet e_j.user = u\} \wedge T_{ref} = \mathcal{L} \setminus T_{test})\}$	Partition the complete trace set to compare a single support attendant with other attendants;
	$\mathcal{P}^a$	$(\mathcal{R}^{\text{suspended}}_{=?}[\text{F End}], \alpha)$	Evaluate the "suspended" reward at a confidence level of $(1-\alpha)$;
	CP	$(>)$	Compare the results to see if a support user is more likely to suspend tickets;
	CR	$\forall u \in U^T \bullet perm'(u) = (perm(u) \setminus \{\text{Suspend}\}) \cup \{\text{SuspendWithApproval}\}$	If so then suspension should require approval.
P2.2	*filter*	$\{(T_{test}, T_{ref}) \in \mathbb{P}\mathcal{L}\times\mathbb{P}\mathcal{L} \mid \exists u^c \in Client \wedge \exists u^s \in U^T \bullet (T_{test} = \{(id, \langle e_1, e_2, \dots e_N\rangle) \in \mathcal{L}\} \mid \exists 1 \leq j \leq N \bullet e_j.user = u^c \wedge \exists 1 \leq k \leq N \bullet e_k.user = u^s\} \wedge T_{ref} = \mathcal{L} \setminus T_{test})\}$	Partition the log to examine those clients identified in **P2.1**. For each client the test set is those traces which involve the identified support attendant and the reference set is all other users;
	$\mathcal{P}^a$	$(\mathcal{R}^{\text{suspended}}_{=?}[\text{F End}], \alpha)$	Evaluate the "suspended" reward at a confidence level of $(1-\alpha)$;
	CP	$(<)$	Compare the results to see if the client is less likely to be suspended if serviced by other support attendants;
	CR	$\forall u^c \in U^T \bullet perm'(u^c) = \emptyset$	If so then remove all access permissions from the user.
P3	*filter*	$\{(T_{test}, T_{ref}) \in \mathbb{P}\mathcal{L}\times\mathbb{P}\mathcal{L} \mid \exists u \in Support \bullet (T_{test} = \{(id, \langle e_1, e_2, \dots e_N\rangle) \in \mathcal{L} \mid \exists 1 \leq j \leq N \bullet e_j.user = u\} \wedge T_{ref} = \mathcal{L} \setminus T_{test})\}$	Partition the trace data to compare a single support attendants to all other attendants;
	$\mathcal{P}^a$	$(\mathcal{P}_{=?}[\neg\text{Suspended U Cancelled}], \alpha)$	Evaluate the probability of cancelling a ticket without first suspending it;
	CP	$(>)$	Compare the results to see if the support attendant does this more often than normal;
	CR	$\forall u \in U^T \bullet perm'(u) = (perm(u) \setminus \{\text{Cancel}\}) \cup \{\text{CancelWithApproval}\}$	If so then they require approval to cancel tickets.
P4	*filter*	$\{(T_{test}, T_{ref}) \in \mathbb{P}\mathcal{L}\times\mathbb{P}\mathcal{L} \mid \exists u \in Support \bullet (T_{test} = \{(id, \langle e_1, e_2, \dots e_N\rangle) \in \mathcal{L} \mid \exists 1 \leq j \leq N \bullet e_j.user = u\} \wedge T_{ref} = \mathcal{L} \setminus T_{test})\}$	Partition the trace data to compare a single support attendants to all other attendants;
	$\mathcal{P}^a$	$(\mathcal{P}_{=?}[\neg\text{Opened U Abandoned}], \alpha)$	Evaluate the probability of abandoning tickets that have been opened on behalf of other users at a confidence level of $(1-\alpha)$;
	CP	$(>)$	Compare the results to see if the support user does this more often than normal;
	CR	$\forall u \in U^T \bullet perm'(u) = perm(u) \setminus \{\text{OpenOnBehalf}\}$	If so then remove their ability to open tickets on behalf of other users.

Continued on next page

Table B.1 – *Continued from previous page*

Policy		Value	Description
P5.1	*filter*	$\{(T_{test}, T_{ref}) \in \mathbb{P}\mathcal{L} \times \mathbb{P}\mathcal{L} \mid \exists u \in Suport \bullet (T_{test} = \{(id, \langle e_1, e_2, \dots e_N \rangle) \in \mathcal{L} \mid \exists 1 \leq j \leq N \bullet e_j.user = u\} \wedge T_{ref} = \mathcal{L} \setminus T_{test})\}$	Partition the trace data to compare a single support attendants to all other attendants;
	$\mathcal{P}^a$	$(\mathcal{R}^{\text{lazy}}_{=?}[\text{F End}], \alpha_1)$	Evaluate the reward associated with "Lazy" support attendants at a confidence level of $(1 - \alpha_1)$;
	CP	$(>)$	Compare the reward for this support attendant with all other attendants;
	CR	$\forall u \in U^T \bullet perm'(u) = (perm(u) \setminus \{\mathsf{Check}, \mathsf{Solve}, \mathsf{Suspend}\}) \cup \{\mathsf{MonitoredCheck}, \mathsf{MonitoredSolve}, \mathsf{MonitoredSuspend}\}$	If they are found to be "Lazy" then monitor their activity.
P5.2	*filter*	$\{(T_{test}, T_{ref}) \in \mathbb{P}\mathcal{L} \times \mathbb{P}\mathcal{L} \mid \exists u \in U^T \bullet (T_{test} = \{(id, \langle e_1, e_2, \dots e_N \rangle) \in \mathcal{L} \mid \exists 1 \leq j \leq N \bullet e_j.user = u\} \wedge T_{ref} = \mathcal{L} \setminus T_{test})\}$	Partition the log to examine the traces of support attendants found in **P5.1**;
	$\mathcal{P}^a$	$(\mathcal{R}^{\text{lazy}}_{=?}[\text{F End}], \alpha_2)$	Evaluate the reward associated with "Lazy" support attendants at a confidence level of $(1 - \alpha_2)$ where $(1 - \alpha_2) > (1 - \alpha_1)$;
	CP	$(>)$	Compare the reward for this support attendant with all others;
	CR	$\forall u \in U^T \bullet perm'(u) = (perm(u) \setminus \{\mathsf{Check}, \mathsf{Solve}, \mathsf{Suspend}\}) \cup \{\mathsf{CheckWithApproval}, \mathsf{SolveWithApproval}, \mathsf{SuspendWithApproval}\}$	If they are found to be "Lazy" then their activity requires approval.
P6.1	*filter*	$\{(T_{test}, T_{ref}) \in \mathbb{P}\mathcal{L} \times \mathbb{P}\mathcal{L} \mid \exists u \in Client \bullet (T_{test} = \{(id, \langle e_1, e_2, \dots e_N \rangle) \in \mathcal{L} \mid \exists 1 \leq j \leq N \bullet e_j.user = u\} \wedge T_{ref} = \mathcal{L} \setminus T_{test})\}$	Partition the trace data to compare a single client user to all other clients;
	$\mathcal{P}^a$	$(\mathcal{R}^{\text{expensive}}_{=?}[\text{F End}], \alpha)$	Evaluate the reward associated with "Expensive" tickets at a confidence level of $(1 - \alpha)$;
	CP	$(>)$	Compare the results to see if the client is more "expensive" than normal;
	CR	$\forall u \in U^T \bullet perm'(u) = (perm(u) \setminus \{\mathsf{Open}, \mathsf{Reopen}, \mathsf{AddInformation}\}) \cup \{\mathsf{MonitoredOpen}, \mathsf{MonitoredReopen}, \mathsf{MonitoredAddInformation}\}$	If so then monitor the client.
P6.2	*filter*	$\{(T_{test}, T_{ref}) \in \mathbb{P}\mathcal{L} \times \mathbb{P}\mathcal{L} \mid \exists u^c \in U^T \wedge \exists u^s \in Support \bullet (T_{test} = \{(id, \langle e_1, e_2, \dots e_N \rangle) \in \mathcal{L}\} \mid \exists 1 \leq j \leq N \bullet e_j.user = u^c \wedge \exists 1 \leq k \leq N \bullet e_k.user = u^s\} \wedge T_{ref} = \mathcal{L} \setminus T_{test})\}$	Partition the log to examine those support attendants who dealt with the clients identified in **P6.1**;
	$\mathcal{P}^a$	$(\mathcal{R}^{\text{expensive}}_{=?}[\text{F End}], \alpha)$	Evaluate the reward associated with "Expensive" tickets at a confidence level of $(1 - \alpha)$;
	CP	$(<)$	Check to see if tickets processed by this support attendant is less "expensive" than those for other attendants;
	CR	$\forall u \in U^T \bullet perm'(u^c) = perm(u^c) \setminus \{\mathsf{Open}\}$	If so then the client should have their ability to open new tickets removed.

List of References

[1] R. Calinescu, K. Johnson, and C. Paterson, "FACT: A probabilistic model checker for formal verification with confidence intervals," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pp. 540–546, Springer Berlin Heidelberg, 2016.

[2] C. E. da Silva, J. D. S. da Silva, C. Paterson, and R. Calinescu, "Self-adaptive role-based access control for business processes," in *Proceedings of the 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2017)*, pp. 193–203, IEEE Press, 2017.

[3] C. Paterson and R. Calinescu, "Accurate analysis of quality properties of software with observation-based Markov chain refinement," in *IEEE International Conference on Software Architecture*, pp. 121–130, 2017.

[4] C. A. Paterson and R. Calinescu, "Observation-enhanced QoS analysis of component-based systems," *IEEE Transactions on Software Engineering*, 2018.

[5] R. Calinescu, K. Johnson, and C. Paterson, "Efficient parametric model checking using domain-specific modelling patterns," in *Proceedings of the 40th International Conference on Software Engineering: New Ideas and Emerging Results*, pp. 61–64, ACM, 2018.

[6] E. M. Clarke, E. A. Emerson, and J. Sifakis, "Model checking: algorithmic verification and debugging," *Communications of the ACM*, vol. 52, no. 11, pp. 74–84, 2009.

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

[8] M. Kwiatkowska, G. Norman, and D. Parker, "Prism 4.0: Verification of probabilistic real-time systems," in *Proceedings 23rd international conference on computer aided verification*, pp. 585–591, Springer, 2011.

[9] J.-P. Katoen, I. S. Zapreev, E. M. Hahn, H. Hermanns, and D. N. Jansen, "The ins and outs of the probabilistic model checker MRMC," *Performance evaluation*, vol. 68, no. 2, pp. 90–104, 2011.

[10] C. Dehnert, S. Junges, N. Jansen, F. Corzilius, M. Volk, H. Bruintjes, J.-P. Katoen, and E. Abraham, "Prophesy: A probabilistic parameter synthesis tool," in *International Conference on Computer Aided Verification*, pp. 214–231, Springer, 2015.

[11] C. Dehnert, S. Junges, J.-P. Katoen, and M. Volk, "A STORM is coming: A modern probabilistic model checker," in *International Conference on Computer Aided Verification*, pp. 592–600, Springer, 2017.

[12] M. K. Haider, A. K. Ismail, and I. A. Qazi, "Markovian models for electrical load prediction in smart buildings," in *Neural Information Processing*, pp. 632–639, Springer, 2012.

[13] T. Mendt, C. Sinz, and O. Tveretina, "Probabilistic model checking of constraints in a supply chain business process," in *Business Information Systems*, pp. 1–12, Springer, 2011.

[14] A. Cerone, P. A. Lindsay, and S. Connelly, "Formal analysis of human-computer interaction using model-checking," in *Software Engineering and Formal Methods, 2005. SEFM 2005. Third IEEE International Conference on*, pp. 352–361, IEEE, 2005.

[15] T. T. B. Hanh and D. Van Hung, "Verification of an air-traffic control system with probabilistic real-time model-checking," *UNU-IIST report*, 2007.

[16] L. E. M. Morales, "Business process verification using a formal compositional approach and timed automata," in *Computing Conference (CLEI), 2013 XXXIX Latin American*, pp. 1–10, IEEE, 2013.

[17] F. Corradini, A. Polini, A. Polzonetti, and B. Re, "Business processes verification for e-government service delivery," *Information Systems Management*, vol. 27, no. 4, pp. 293–308, 2010.

[18] T. H. Davenport, *Process innovation: reengineering work through information technology*. Harvard Business Press, 1993.

[19] D. Barnes, *Understanding business: processes*. Psychology Press, 2001.

[20] L. Galloway, F. Rowbotham, and M. Azhashemi, *Operations management in context*. Routledge, 2012.

[21] H. Schonenberg, R. Mans, N. Russell, N. Mulyar, and W. van der Aalst, "Process flexibility: A survey of contemporary approaches," in *Advances in Enterprise Engineering I*, pp. 16–30, Springer, 2008.

[22] S. Sadiq, W. Sadiq, and M. Orlowska, "Pockets of flexibility in workflow specification," in *Conceptual ModelingER 2001*, pp. 513–526, Springer, 2001.

[23] R. Calinescu, C. Ghezzi, K. Johnson, M. Pezze, Y. Rafiq, and G. Tamburrelli, "Formal verification with confidence intervals to establish quality of service properties of software systems," *IEEE Transactions on Reliability*, vol. PP, pp. 1–16, August 2015.

[24] E. M. Clarke and E. A. Emerson, "Design and synthesis of synchronization skeletons using branching time temporal logic," in *Workshop on Logic of Programs*, pp. 52–71, Springer, 1981.

[25] J.-P. Queille and J. Sifakis, "Specification and verification of concurrent systems in cesar," in *International Symposium on programming*, pp. 337–351, Springer, 1982.

[26] C. Baier and J.-P. Katoen, *Principles of model checking*. MIT press, 2008.

[27] M. Kwiatkowska, G. Norman, and D. Parker, "Stochastic model checking," in *International School on Formal Methods for the Design of Computer, Communication and Software Systems*, pp. 220–270, Springer, 2007.

[28] H. Hansson and B. Jonsson, "A logic for reasoning about time and reliability," *Formal aspects of computing*, vol. 6, no. 5, pp. 512–535, 1994.

[29] E. M. Clarke, E. A. Emerson, and A. P. Sistla, "Automatic verification of finite-state concurrent systems using temporal logic specifications," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 8, no. 2, pp. 244–263, 1986.

[30] N. Sato and K. S. Trivedi, *Stochastic modeling of composite web services for closed-form analysis of their performance and reliability bottlenecks*. Springer, 2007.

[31] S. Gallotti, C. Ghezzi, R. Mirandola, and G. Tamburrelli, "Quality prediction of service compositions through probabilistic model checking," in *Quality of Software Architectures. Models and Architectures*, pp. 119–134, Springer, 2008.

[32] S. Gerasimou, G. Tamburrelli, and R. Calinescu, "Search-based synthesis of probabilistic models for quality-of-service software engineering (t)," in *Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference on*, pp. 319–330, Nov 2015.

[33] 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.

[34] R. Calinescu, S. Gerasimou, and A. Banks, "Self-adaptive software with decentralised control loops," in *International Conference on Fundamental Approaches to Software Engineering*, pp. 235–251, Springer, 2015.

[35] W. Whitt, "Continuous-time Markov chains." `http://www.columbia.edu/~ww2040/6711F14/CTMCnotes121312.pdf`, (accessed on 6/11/2014).

[36] A. Aziz, K. Sanwal, V. Singhal, and R. Brayton, "Verifying continuous time Markov chains," in *Computer Aided Verification*, pp. 269–276, Springer, 1996.

[37] C. Baier, J.-P. Katoen, and H. Hermanns, "Approximative symbolic model checking of continuous-time Markov chains," in *CONCUR99 Concurrency Theory*, pp. 146–161, Springer, 1999.

[38] C. Baier, B. Haverkort, H. Hermanns, and J. P. Katoen, "Model-checking algorithms for continuous-time Markov chains," *IEEE Transactions on Software Engineering*, vol. 29, pp. 524–541, June 2003.

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

[40] M. Fackrell, "Modelling healthcare systems with phase-type distributions," *Health care management science*, vol. 12, no. 1, pp. 11–26, 2009.

[41] E. Ishay, *Fitting phase-type distributions to data from a telephone call center*. PhD thesis, Technion-Israel Institute Of Technology, 2002.

[42] A. H. Marshall and M. Zenga, "Simulating Coxian phase-type distributions for patient survival," *International Transactions in Operational Research*, vol. 16, no. 2, pp. 213–226, 2009.

[43] 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.

[44] L. Korenčiak, J. Krčál, and V. Řehák, "Dealing with zero density using piecewise phase-type approximation," in *Computer Performance Eng.*, pp. 119–134, Springer, 2014.

[45] A. David and S. Larry, "The least variable phase type distribution is erlang," *Stochastic Models*, vol. 3, no. 3, pp. 467–473, 1987.

[46] Y. Fang and I. Chlamtac, "Teletraffic analysis and mobility modeling of pcs networks," *IEEE Transactions on Communications*, vol. 47, no. 7, pp. 1062–1072, 1999.

[47] Y. Fang, "Hyper-Erlang distribution model and its application in wireless mobile networks," *Wireless Networks*, vol. 7, no. 3, pp. 211–219, 2001.

[48] E. M. Clarke, E. A. Emerson, and A. P. Sistla, "Automatic verification of finite state concurrent system using temporal logic specifications: A practical approach," in *Proceedings of the 10th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, POPL '83, (New York, NY, USA), pp. 117–126, ACM, 1983.

[49] E. M. Clarke, E. A. Emerson, and A. P. Sistla, "Automatic verification of finite-state concurrent systems using temporal logic specifications," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 8, no. 2, pp. 244–263, 1986.

[50] C. M. University, "The SMV system," 2007. `http://www.cs.cmu.edu/~modelcheck/smv.html` (accessed on 6/11/2014).

[51] G. J. Holzmann, "The model checker SPIN," *IEEE Transactions on software engineering*, vol. 23, no. 5, pp. 279–295, 1997.

[52] M. Kwiatkowska and D. Parker, "Advances in probabilistic model checking," *Software Safety and Security: Tools for Analysis and Verification*, vol. 33, no. 126, 2012. Editors Grumberg, O., Nipkow, T. and Esparza, J.

[53] 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 *Hardware and Software: Verification and Testing*, pp. 69–85, Springer, 2007.

[54] M. Kwiatkowska, G. Norman, and D. Parker, "PRISM: Probabilistic symbolic model checker," in *Computer performance evaluation: modelling techniques and tools*, pp. 200–204, Springer, 2002.

[55] M. Kwiatkowska, G. Norman, and D. Parker, "Advances and challenges of probabilistic model checking," in *Communication, Control, and Computing (Allerton), 2010 48th Annual Allerton Conference on*, pp. 1691–1698, IEEE, 2010.

[56] G. T. Leavens and M. Sitaraman, *Foundations of component-based systems*. Cambridge University Press, 2000.

[57] A. Aleti, B. Buhnova, L. Grunske, A. Koziolek, and I. Meedeniya, "Software architecture optimization methods: A systematic literature review," *IEEE Transactions on Software Engineering*, vol. 39, no. 5, pp. 658–683, 2013.

[58] S. Becker, L. Grunske, R. Mirandola, and S. Overhage, "Performance prediction of component-based systems," in *Architecting Systems with Trustworthy Components: International Seminar, Dagstuhl Castle, Germany, December 12-17, 2004. Revised Selected Papers* (R. H. Reussner, J. A. Stafford, and C. A. Szyperski, eds.), (Berlin, Heidelberg), pp. 169–192, Springer Berlin Heidelberg, 2006.

[59] H. Koziolek, "Performance evaluation of component-based software systems: A survey," *Performance Evaluation*, vol. 67, no. 8, pp. 634 – 658, 2010. Special Issue on Software and Performance.

[60] A. Koziolek, D. Ardagna, and R. Mirandola, "Hybrid multi-attribute QoS optimization in component based software systems," *Journal of Systems and Software*, vol. 86, no. 10, pp. 2542–2558, 2013.

[61] M. A. Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis, *Modelling with generalized stochastic Petri nets*. John Wiley & Sons, Inc., 1994.

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

[63] 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.

[64] J. A. Carrasco, "Computationally efficient and numerically stable reliability bounds for repairable fault-tolerant systems," *Computers, IEEE Transactions on*, vol. 51, no. 3, pp. 254–268, 2002.

[65] A. Hessel, K. G. Larsen, M. Mikucionis, B. Nielsen, P. Pettersson, and A. Skou, "Testing real-time systems using uppaal," in *Formal Methods and Testing: An Outcome of the FORTEST Network, Revised Selected Papers* (R. M. Hierons, J. P. Bowen, and M. Harman, eds.), (Berlin, Heidelberg), pp. 77–117, Springer Berlin Heidelberg, 2008.

[66] S. Becker *et al.*, "The Palladio component model for model-driven performance prediction," *J. Syst. Softw.*, vol. 82, no. 1, pp. 3–22, 2009.

[67] S. Baarir, M. Beccuti, D. Cerotti, M. De Pierro, S. Donatelli, and G. Franceschinis, "The GreatSPN tool: recent enhancements," *ACM SIGMETRICS Performance Evaluation Review*, vol. 36, no. 4, pp. 4–9, 2009.

[68] G. Behrmann, A. David, K. G. Larsen, J. Hakansson, P. Petterson, W. Yi, and M. Hendriks, "Uppaal 4.0," in *Third International Conference on the Quantitative Evaluation of Systems - (QEST'06)*, pp. 125–126, Sept 2006.

[69] R. Calinescu, K. Johnson, and Y. Rafiq, "Developing self-verifying service-based systems," in *Automated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference on*, pp. 734–737, Nov 2013.

[70] C. Ghezzi, M. Pezzè, M. Sama, and G. Tamburrelli, "Mining behavior models from user-intensive web applications," in *Proceedings of the 36th International Conference on Software Engineering*, pp. 277–287, ACM, 2014.

[71] G. Su and D. S. Rosenblum, "Asymptotic bounds for quantitative verification of perturbed probabilistic systems," in *International Conference on Formal Engineering Methods*, pp. 297–312, Springer, 2013.

[72] A. Bobbio and A. Cumani, "Ml estimation of the parameters of a ph distribution in triangular canonical form," *Computer performance evaluation*, vol. 22, pp. 33–46, 1992.

[73] 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.

[74] A. Horvath and M. Telek, "Approximating heavy tailed behaviour with phase type distributions," 2000. `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.26.6351`.

[75] 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*, pp. 49–75, Springer, 2016.

[76] A. Bobbio, A. Horváth, and M. Telek, "Matching three moments with minimal acyclic phase type distributions," *Stochastic Models*, vol. 21, no. 2–3, pp. 303–326, 2005.

[77] 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.

[78] S. Asmussen, O. Nerman, and M. Olsson, "Fitting phase-type distributions via the EM algorithm," *Scandinavian Journal of Statistics*, pp. 419–441, 1996.

[79] 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, pp. 245–258, July 2006.

[80] 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.

[81] 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.

[82] 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.

[83] P. Reinecke, T. Krauss, and K. Wolter, "Hyperstar: Phase-type fitting made easy," in *2012 Ninth International Conference on Quantitative Evaluation of Systems*, pp. 201–202, IEEE, 2012.

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

[85] P. Karmakar and K. Gopinath, "Are Markov models effective for storage reliability modelling?," *arXiv preprint arXiv:1503.07931*, 2015.

[86] G. Ciobanu and A. S. Rotaru, "Phase: a stochastic formalism for phase-type distributions," in *Formal Methods and Software Engineering*, pp. 91–106, Springer, 2014.

[87] T. Brázdil, L. Korenčiak, J. Krčál, P. Novotnỳ, and V. Řehák, "Optimizing performance of continuous-time stochastic systems using timeout synthesis," in *Quantitative Evaluation of Systems*, pp. 141–159, Springer, 2015.

[88] 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.

[89] G. G. I. López, H. Hermanns, and J.-P. Katoen, *Beyond Memoryless Distributions: Model Checking Semi-Markov Chains*, pp. 57–70. Springer, 2001.

[90] E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith, "Progress on the state explosion problem in model checking," in *Informatics*, pp. 176–194, Springer, 2001.

[91] E. A. Emerson and E. M. Clarke, "Using branching time temporal logic to synthesize synchronization skeletons," *Science of Computer programming*, vol. 2, no. 3, pp. 241–266, 1982.

[92] E. M. Clarke, O. Grumberg, and D. E. Long, "Model checking and abstraction," *ACM transactions on Programming Languages and Systems (TOPLAS)*, vol. 16, no. 5, pp. 1512–1542, 1994.

[93] E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith, "Counterexample-guided abstraction refinement," in *International Conference on Computer Aided Verification*, pp. 154–169, Springer, 2000.

[94] H. Hermanns, B. Wachter, and L. Zhang, "Probabilistic cegar," in *International Conference on Computer Aided Verification*, pp. 162–175, Springer, 2008.

[95] H. Peng and S. Tahar, "A survey on compositional verification," *Department of Electrical and Computer Engineering, Concordia Univ*, 1998.

[96] A. Pnueli, "In transition from global to modular temporal reasoning about programs," in *Logics and models of concurrent systems*, pp. 123–144, Springer, 1985.

[97] E. M. Clarke, D. E. Long, and K. L. McMillan, "Compositional model checking," in *[1989] Proceedings. Fourth Annual Symposium on Logic in Computer Science*, pp. 353–362, IEEE, 1989.

[98] O. Grumberg and D. E. Long, "Model checking and modular verification," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 16, no. 3, pp. 843–871, 1994.

[99] Y. Kesten and A. Pnueli, "A compositional approach to ctl* verification," *Theoretical Computer Science*, vol. 331, no. 2-3, pp. 397–428, 2005.

[100] M. Kwiatkowska, G. Norman, D. Parker, and H. Qu, "Assume-guarantee verification for probabilistic systems," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pp. 23–37, Springer, 2010.

[101] R. Calinescu, S. Kikuchi, and K. Johnson, "Compositional reverification of probabilistic safety properties for large-scale complex it systems," in *Monterey Workshop*, pp. 303–329, Springer, 2012.

[102] P. Ashok, Y. Butkova, H. Hermanns, and J. Křetínskỳ, "Continuous-time Markov decisions based on partial exploration," *arXiv preprint arXiv:1807.09641*, 2018.

[103] G. Norman and D. Parker, "Quantitative verification: Formal guarantees for timeliness, reliability and performance," 2014.

[104] W. L. Oberkampf, S. M. DeLand, B. M. Rutherford, K. V. Diegert, and K. F. Alvin, "Error and uncertainty in modeling and simulation," *Reliability Engineering & System Safety*, vol. 75, no. 3, pp. 333–357, 2002.

[105] R. Calinescu, Y. Rafiq, K. Johnson, and M. E. Bakır, "Adaptive model learning for continual verification of non-functional properties," in *Proceedings of the 5th ACM/SPEC international conference on Performance engineering*, pp. 87–98, ACM, 2014.

[106] T. W. Anderson and L. A. Goodman, "Statistical inference about Markov chains," *The Annals of Mathematical Statistics*, pp. 89–110, 1957.

[107] J. Neyman, "Xoutline of a theory of statistical estimation based on the classical theory of probability," *Phil. Trans. R. Soc. Lond. A*, vol. 236, no. 767, pp. 333–380, 1937.

[108] I. Miller and M. Miller, *John E. Freund's mathematical statistics with applications*, vol. 7. Prentice Hall Upper Saddle River,, NJ, 2004.

[109] J. M. Utts, *Seeing through statistics*. Cengage Learning, 2014.

[110] G. Cumming, "The new statistics: Why and how," *Psychological science*, vol. 25, no. 1, pp. 7–29, 2014.

[111] C. P. Sison and J. Glaz, "Simultaneous confidence intervals and sample size determination for multinomial proportions," *Journal of the American Statistical Association*, vol. 90, no. 429, pp. 366–369, 1995.

[112] A. Filieri, C. Ghezzi, and G. Tamburrelli, "Run-time efficient probabilistic model checking," in *Proceedings of the 33rd international conference on software engineering*, pp. 341–350, ACM, 2011.

[113] N. Jansen, F. Corzilius, M. Volk, R. Wimmer, E. Ábrahám, J.-P. Katoen, and B. Becker, "Accelerating parametric probabilistic verification," in *International Conference on Quantitative Evaluation of Systems*, pp. 404–420, Springer, 2014.

[114] C. Daws, "Symbolic and parametric model checking of discrete-time Markov chains," in *International Colloquium on Theoretical Aspects of Computing*, pp. 280–294, Springer, 2004.

[115] E. M. Hahn, H. Hermanns, B. Wachter, and L. Zhang, "PARAM: A model checker for parametric Markov models," in *CAV'10*, pp. 660–664, 2010.

[116] L. A. Goodman, "On simultaneous confidence intervals for multinomial proportions," *Technometrics*, vol. 7, no. 2, pp. 247–254, 1965.

[117] K.-S. Kwong and B. Iglewicz, "On singular multivariate normal distribution and its applications," *Computational statistics & data analysis*, vol. 22, no. 3, pp. 271–285, 1996.

[118] C. P. Quesenberry and D. Hurst, "Large sample simultaneous confidence intervals for multinomial proportions," *Technometrics*, vol. 6, no. 2, pp. 191–195, 1964.

[119] C. Dehnert, S. Junges, N. Jansen, F. Corzilius, M. Volk, H. Bruintjes, J.-P. Katoen, and E. Ábrahám, "PROPhESY: A PRObabilistic ParamEter SYnthesis Tool," in *International Conference on Computer Aided Verification*, pp. 214–231, 2015.

[120] C. Chatfield, "Statistical inference regarding Markov chain models," *Applied Statistics*, pp. 7–20, 1973.

[121] O. Andrei, M. Calder, M. Higgs, and M. Girolami, "Probabilistic model checking of dtmc models of user activity patterns," in *International Conference on Quantitative Evaluation of Systems*, pp. 138–153, Springer, 2014.

[122] T. Zheng, C. M. Woodside, and M. Litoiu, "Performance model estimation and tracking using optimal filters," *IEEE Transactions on software engineering*, vol. 34, no. 3, pp. 391–406, 2008.

[123] I. Epifani, C. Ghezzi, R. Mirandola, and G. Tamburrelli, "Model evolution by run-time parameter adaptation," in *Proceedings of the 31st International Conference on Software Engineering*, pp. 111–121, IEEE Computer Society, 2009.

[124] I. Epifani, C. Ghezzi, and G. Tamburrelli, "Change-point detection for black-box services," in *Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering*, pp. 227–236, ACM, 2010.

[125] R. Calinescu, K. Johnson, and Y. Rafiq, "Using observation ageing to improve Markovian model learning in QoS engineering," in *Proceedings of the 2nd ACM/SPEC International Conference on Performance engineering*, pp. 505–510, ACM, 2011.

[126] I. O. Kozine and L. V. Utkin, "Interval-valued finite Markov chains," *Reliable computing*, vol. 8, no. 2, pp. 97–113, 2002.

[127] D. Škulj, "Discrete time Markov chains with interval probabilities," *International journal of approximate reasoning*, vol. 50, no. 8, pp. 1314–1329, 2009.

[128] T. Chen, T. Han, and M. Kwiatkowska, "On the complexity of model checking interval-valued discrete time Markov chains," *Information Processing Letters*, vol. 113, no. 7, pp. 210–216, 2013.

[129] A. Puggelli, W. Li, A. L. Sangiovanni-Vincentelli, and S. A. Seshia, "Polynomial-time verification of PCTL properties of mdps with convex uncertainties," in *International Conference on Computer Aided Verification*, pp. 527–542, Springer, 2013.

[130] A. Legay, B. Delahaye, and S. Bensalem, "Statistical model checking: An overview," in *International conference on runtime verification*, pp. 122–135, Springer, 2010.

[131] H. L. Younes, "Ymer: A statistical model checker," in *International Conference on Computer Aided Verification*, pp. 429–433, Springer, 2005.

[132] K. Sen, M. Viswanathan, and G. Agha, "Statistical model checking of black-box probabilistic systems," in *International Conference on Computer Aided Verification*, pp. 202–215, Springer, 2004.

[133] J.-P. Katoen and I. S. Zapreev, "Simulation-based CTMC model checking: An empirical evaluation," in *2009 Sixth International Conference on Quantitative Evaluation of Systems*, pp. 31–40, IEEE, 2009.

[134] G. Agha and K. Palmskog, "A survey of statistical model checking," *ACM Trans. Model. Comput. Simul.*, vol. 28, pp. 6:1–6:39, Jan. 2018.

[135] P. A. Legg, O. Buckley, M. Goldsmith, and S. Creese, "Caught in the act of an insider attack: detection and assessment of insider threat," in *2015 IEEE International Symposium on Technologies for Homeland Security (HST)*, pp. 1–6, April 2015.

[136] E. Cole, "Insider threats and the need for fast and directed response," tech. rep., SANS Institute InfoSec Reading Room, 2015.

[137] G. Silowash, D. Cappelli, A. Moore, R. Trzeciak, T. Shimeall, and L. Flynn, "Common sense guide to mitigating insider threats," Tech. Rep. CMU/SEI-2012-TR-012, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 2012.

[138] D. M. Cappelli, A. P. Moore, and R. F. Trzeciak, *The CERT Guide to Insider Threats: How to Prevent, Detect, and Respond to Information Technology Crimes*. Addison-Wesley Professional, 1st ed., 2012.

[139] International Organization for Standardization, "ISO/IEC 27005:2011: Information technology – Security techniques – Information security risk management," 2011.

[140] ANSI, "Role based access control," Tech. Rep. ANSI INCITS 359-2004, NIST, 2004.

[141] C. Bailey, D. W. Chadwick, and R. de Lemos, "Self-adaptive federated authorization infrastructures," *Journal of Computer and System Sciences*, vol. 80, no. 5, pp. 935 – 952, 2014.

[142] M. B. Salem, S. Hershkop, and S. J. Stolfo, "A survey of insider attack detection research," in *Insider Attack and Cyber Security*, pp. 69–90, Springer, 2008.

[143] R. Bharadwaj and C. Heitmeyer, "Applying the scr requirements specification method to practical systems: A case study," tech. rep., NAVAL RESEARCH LAB WASHINGTON DC CENTER FOR HIGH ASSURANCE COMPUTING SYSTEMS (CHACS), 1996.

[144] R. W. McGraw, "Risk adaptable access control (RAdAC)," tech. rep., National Institute of Standards and Technology, McLean and Clifton, VA, United States, 2009.

[145] R. A. Shaikh, K. Adi, and L. Logrippo, "Dynamic risk-based decision methods for access control systems," *Computers & Security*, vol. 31, no. 4, pp. 447–464, 2012.

[146] P. C. Cheng, P. Rohatgi, C. Keser, P. Karger, G. Wagner, and A. Reninger, "Fuzzy multi-level security: An experiment on quantified risk-adaptive access control," in *IEEE Symposium on Security and Privacy (SP '07)*, pp. 222–230, 2007.

[147] S. Kandala, R. Sandhu, and V. Bhamidipati, "An attribute based framework for risk-adaptive access control models," in *Sixth International Conference on Availability, Reliability and Security (ARES)*, pp. 236–241, Aug 2011.

[148] J. Park and R. Sandhu, "The uconabc usage control model," *ACM Transactions on Information and System Security (TISSEC)*, vol. 7, pp. 128–174, feb 2004.

[149] K. Z. Bijon, R. Krishnan, and R. Sandhu, "A framework for risk-aware role based access control," in *Proceedings 6th IEEE-CNS Symposium on Security Analytics and Automation (SAFECONFIG)*, pp. 462–469, Oct 2013.

[150] D. W. Chadwick *et al.*, "PERMIS: A Modular Authorization Infrastructure," *Concurrency and Computation: Practice and Experience*, vol. 20, pp. 1341–1357, Aug. 2008.

[151] C. Tsigkanos, L. Pasquale, C. Menghi, C. Ghezzi, and B. Nuseibeh, "Engineering topology aware adaptive security: Preventing requirements violations at runtime," in *IEEE 22nd International Requirements Engineering Conference (RE)*, Aug 2014.

[152] C. Tsigkanos, L. Pasquale, C. Ghezzi, and B. Nuseibeh, "Ariadne: Topology aware adaptive security for cyber-physical systems," in *IEEE/ACM 37th IEEE International Conference on Software Engineering (ICSE 2015)*, vol. 2, pp. 729–732, May 2015.

[153] L. Pasquale, C. Ghezzi, C. Menghi, C. Tsigkanos, and B. Nuseibeh, "Topology aware adaptive security," in *Proceedings of the 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2014)*, ACM, 2014.

[154] I. Agrafiotis, J. R. Nurse, O. Buckley, P. Legg, S. Creese, and M. Goldsmith, "Identifying attack patterns for insider threat detection," *Computer Fraud & Security*, vol. 2015, no. 7, pp. 9 – 17, 2015.

[155] P. A. Legg, O. Buckley, M. Goldsmith, and S. Creese, "Automated insider threat detection system using user and role-based profile assessment," *IEEE Systems Journal*, no. 99, pp. 1–10, 2015.

[156] S. Iannucci and S. Abdelwahed, "A probabilistic approach to autonomic security management," in *2016 IEEE International Conference on Autonomic Computing (ICAC)*, 2016.

[157] N. Schenker and J. F. Gentleman, "On judging the significance of differences by examining the overlap between confidence intervals," *The American Statistician*, vol. 55, no. 3, pp. 182–186, 2001.

[158] D. Garlan, S.-W. Cheng, A. C. Huang, B. Schmerl, and P. Steenkiste, "Rainbow: architecture-based self-adaptation with reusable infrastructure," *Computer*, vol. 37, pp. 46–54, Oct 2004.

[159] R. Calinescu, "Model-driven autonomic architecture," in *Fourth International Conference on Autonomic Computing (ICAC'07)*, pp. 9–9, June 2007.

[160] J. Kramer and J. Magee, "Self-managed systems: an architectural challenge," in *Future of Software Engineering, 2007. FOSE '07*, pp. 259–268, May 2007.

[161] R. Calinescu, "Implementation of a generic autonomic framework," in *Fourth International Conference on Autonomic and Autonomous Systems (ICAS'08)*, pp. 124–129, March 2008.

[162] E. M. Hahn, H. Hermanns, and L. Zhang, "Probabilistic reachability for parametric markov models," *International Journal on Software Tools for Technology Transfer*, vol. 13, no. 1, pp. 3–19, 2011.

[163] K. Kireyev, L. Palen, and K. Anderson, "Applications of topics models to analysis of disaster-related twitter data," in *NIPS Workshop on Applications for Topic Models: Text and Beyond*, vol. 1, Canada: Whistler, 2009.

[164] B. Robinson, R. Power, and M. Cameron, "An evidence based earthquake detector using twitter," in *Proceedings of the Workshop on Language Processing and Crisis Information 2013*, pp. 1–9, 2013.

[165] N. Chambers, B. Fry, and J. McMasters, "Detecting denial-of-service attacks from social media text: Applying nlp to computer security," in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, vol. 1, pp. 1626–1635, 2018.