# An Approach to Improve Accuracy in Probabilistic Models using State Refinement

Paulo H. Maia, Jeff Kramer, Sebastian Uchitel
Department of Computing
Imperial College London
180 Queen's Gate
London SW7 2AZ, UK
{pmaia,su2}@doc.ic.ac.uk,
j.kramer@imperial.ac.uk

Nabor C. Mendonça
Metrado em Informática Aplicada
Universidade de Fortaleza
Av. Washington Soares, 1321
60811-905 Fortaleza, CE, Brazil
nabor@unifor.br

## ABSTRACT

Probabilistic models are useful in the analysis of system behaviour and non-functional properties. Reliable estimates and measurements of probabilities are needed to annotate behaviour models in order to generate accurate predictions. However, this may not be sufficient, and may still lead to inaccurate results when the system model does not properly reflect the probabilistic choices made by the environment. Thus, not only should the probabilities be accurate in properly reflecting reality, but also the model that is being used. In this paper we propose state refinement as a technique to mitigate this problem, showing that it is guaranteed to preserve or increase the accuracy of the initial model. We present a framework for iteratively improving the accuracy of a probabilistically annotated behaviour model with respect to a set of benchmark properties through iterative state refinements.

## Categories and Subject Descriptors

D.2.4 [**Software Engineering**]: Software / Program Verification—*model checking, statistical methods, verification.*

## General Terms

Design, Languages, Theory, Verification.

## Keywords

Probabilistic model checking, behaviour model, accuracy, refinement.

## 1. INTRODUCTION

Behaviour modelling is a powerful technique which allows developers to describe abstractly and reason about the intended behaviour of a software system. It has the benefit of easing the understanding and analysis of how software systems are expected to behave at runtime, thereby proving useful in uncovering flaws at design time [17]. In addition, behaviour models provide the basis to mechanically check whether a system satisfies some desired set of properties [8, 6].

While most existing approaches for behaviour analysis have been focused on the validation of functional features (for instance, verifying safety properties), there is a growing interest in the integration of quantitative validation into the analysis process to meet non-functional requirements or to quantify the likelihood of properties [14, 3]. In order to allow quantitative evaluation behaviour models have to be augmented with features such as real-time delays [1, 23], cost and reward [16], and probabilities [5, 4, 20, 9].

System behaviour is often modelled using a finite state machine based formalism, such as Labelled Transition Systems (LTS) [13], where each model refers to a system component which interacts with the other components through shared events. Probabilistic models can extend the classical LTS formalism, or variations of it, with probabilistic transitions. Formal verification of such models typically involves checking quantitative properties, defined using a temporal logic language, such as PCTL, and can be carried out via simulation, numerical analysis or by using a probabilistic model checker, such as PRISM [15]. Examples of successful applications of probabilistic model-checking in different domains using PRISM can be found in [16].

There is a significant body of work on probabilistic modelling formalisms, their properties and efficient algorithms for analysing them. However, the problem of constructing probabilistic behaviour models has received less attention. To aid engineers in building probabilistic behaviour models, some approaches (e.g., [7, 19, 21, 2]) assume the existence of a non-probabilistic behaviour model and develop techniques that annotate transitions with probabilities based on additional information such as operational profiles [18], execution traces, and probability estimations. Such kinds of approaches have been used to probabilistically reason about system behaviour in a number of software-related activities, including system reliability prediction [7, 19], model-based testing [21], and information retrieval[2].

Regardless of the annotation techniques, these approaches consider the quality of the information (or lack thereof) used to generate the annotations as the main threat to the accu-

racy of the resulting annotated model. For instance, the reliability of estimations and the size of the execution trace sample space will impact the accuracy of the annotated model.

However, in this paper we show that *even when we use an accurate information source to compute probabilities for annotations, and have a model that captures precisely the non-probabilistic behaviour of the system, inaccurate results can be obtained due to the structure of the non-probabilistic model being annotated.* The essence of the problem is that the annotation processes are heavily influenced by the structure of the model being annotated and may produce different results when annotating non-probabilistic behaviour models that are behaviourally equivalent but structurally different. In summary, the structure of the model to be annotated matters, since the probabilistic information may be misrepresented after the annotation is completed, which may cause the analysis to produce inaccurate predictions (false positive or false negative results on probabilistic properties)

In particular, we show that the notion of state refinement can explain the phenomenon of inaccurate probabilistically annotated behaviour models from accurate probabilistic information and that it can be used to mitigate this problem. We show that state refinement is guaranteed to preserve or increase the accuracy of the initial annotated model. Based on this, we present a framework for iteratively improving the accuracy of a probabilistically-annotated behaviour model with respect to a set of benchmark properties through iterative state refinement of the model-to-be-annotated.

The rest of the paper is organised as follows. Section 2 introduces the motivating system that will be used throughout the paper to illustrate our concepts and techniques. Section 3 contains some definitions of the mathematical foundation we use. The concept of state refinement and its properties is presented in Section 4. Section 5 describes how successive refinements produce an accurate model, while Section 6 presents our proposed framework. A case study conducted using the framework is shown in Section 7. Section 8 compares our work to other related approaches. Finally, conclusions and future work are presented in Section 9.

## 2. MOTIVATING EXAMPLE

Assume we have a set of traces that we will use as information for annotating a provided behaviour model. These traces may have come from a simulation or the observation of the use of an existing system. In this section, we are going to show that depending on the structure of the behaviour model that will be annotated, the result of annotating and model checking may be different from the actual count in the traces.

To conduct these experiments, we took a probabilistic model of a system and generated the set of traces using simulation. More specifically, our behaviour analysis process, illustrated in Figure 1, consists of picking a system behaviour model (BM), which is correct in that it does model the actual behaviour of the system under analysis, and the set of traces generated by a simulation of the system. The simulation is fed by an environment probabilistic behaviour model (EPBM). The traces are input to an algorithm, called Annotator, which annotates the occurrence probabilities of each transition, as observed from the simulation traces, on the provided BM, and generates as output a probabilistically annotated behaviour model (PABM). This PABM is then analysed with respect to a set of desired properties using the PRISM model checker [15]. The process also relies on another algorithm, called Counter, which calculates the probability of each property holding by counting the total number of simulation traces that satisfy that property. Finally, the results obtained with PRISM are compared against those produced by Counter, with their differences, referred to as $\Delta$, indicating the accuracy of the initial BM.

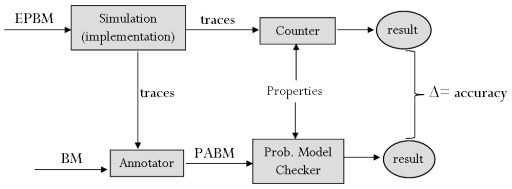


**Figure 1: Behaviour analysis process used in the motivating experiments.**

The running example used in this paper is based on an application called TeleAssistance (TA), a distributed system for medical assistance, originally described in [11]. It consists in a web service process for remote assistance of patients. Once the process is started by the patient, it offers three choices: *sending the patient's vital parameters*, *sending a panic alarm by pressing a button* and *stopping the application.* The first message contains the patient's vital parameters that are forwarded to the Medical Laboratory service (LAB), where the data will be analysed. The LAB replies by sending one of the following results: *change drug*, *change doses* or *send an alarm.* The latter message triggers the intervention of a First-Aid Squad (FAS) composed of doctors, nurses, and paramedics, whose task is to attend to the patient at home in case of emergency. To alert the squad, the process sends an alarm to the FAS. When the patient presses the panic button, the application also generates an alarm sent to the FAS. Finally, the patient may decide to stop the TA service. TA can fail in the following situations: sending an alarm to the FAS, receiving the data analysis from the LAB, or sending a change dose or change drug message to the patient. In all cases, the system goes to a final state indicating that a failure has occured. Figure 2(a) depicts the TA behaviour model.

Let us suppose we are interested in validating the following properties with respect to the TA application:[1]

R1: The probability that no failures ever occurred is greater than 0.7

R2: After a changeDrug or changeDose has occurred, the probability that the next message received by the TA generates an alarm which fails is less then 0.015

R3: The probability that no failures ever occurred and the user has sent the vital parameters or pressed the panic button at least once is greater than 0.65

Our experiment uses two validation scenarios: one using the original TA behaviour model and the other using a behaviourally equivalent but structurally different one. For each scenario we conduct two different validation experiments, as described below.

[1]The first two properties presented here were also originally described in [11], while the third one is an adaptation of other properties described therein.

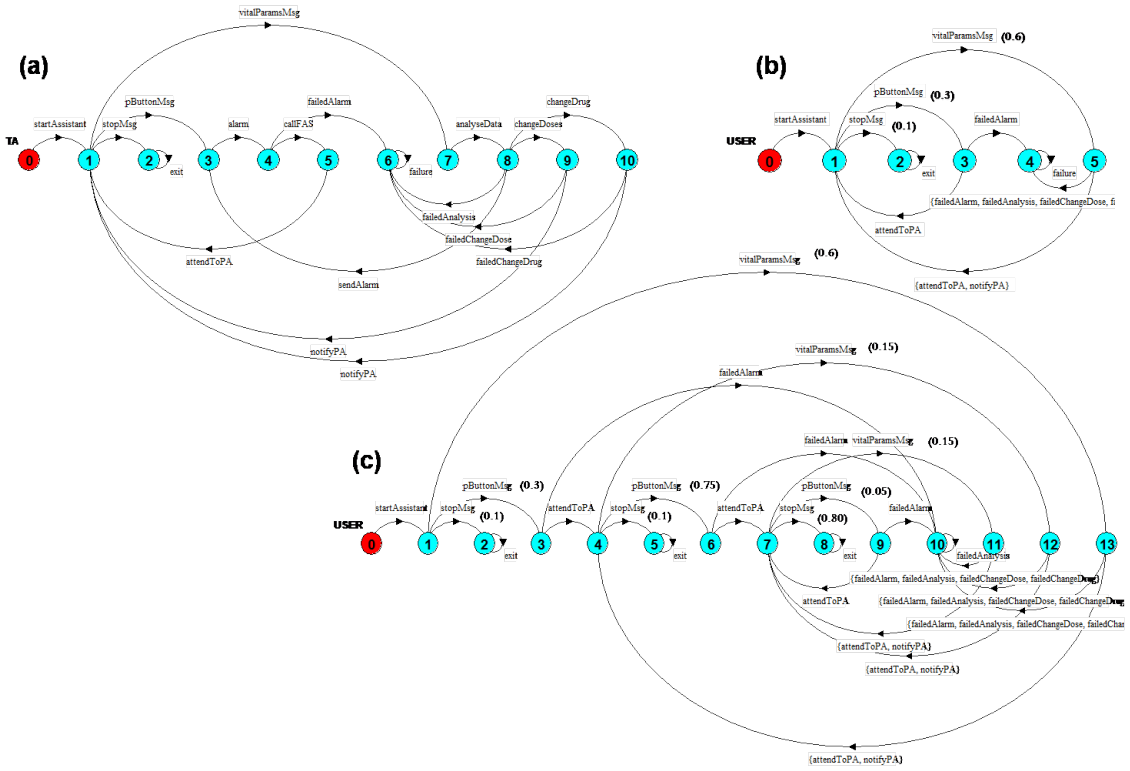


**Figure 2: BM for the TeleAssistance application (a) and the EPBMs used in Experiments 1 and 3 (b), 2 and 4 (c).**

**Table 1: Probabilities of the environment controlled actions in the PABMs of Experiments 1 and 2**

| Action | Experiment 1 | Experiment 2 |
|---|---|---|
| vitalParamMsg | 0.6 | 0.30 |
| pButtonMsg | 0.3 | 0.37 |
| stopMsg | 0.1 | 0.33 |

## 2.1 Using the Provided Behaviour Model

**Experiment 1.** This experiment used the BM shown in Figure 2(a), which is the same one used in [11]. The EPBM is also the same used in that work and it is depicted by Figure 2(b). In that environment model, the user always chooses each of the options provided by the system with the same probability at each interaction cycle. As we can see from Table 1, the probabilities of the environment controlled actions (*vitalParamsMsg*, *pButtonMsg*, and *stopMsg*) in the PABM generated for this experiment represents the same probabilistic choices made by the user.

Table 2 shows the analysis results for this first experiment. We can see that $\Delta$ is small for all three properties analysed, which indicates that the provided BM is accurate with respect to that particular EPBM.

**Experiment 2.** This experiment is similar to the first one, with the exception that we used a different EPBM, shown in Figure 2(c). In that model, the user chooses each option with different probabilities in each of three interactions with the system. We can see from Table 1 that this time the probabilities of the environment controlled actions in the generated PABM are not consistent with the environment probabilistic choices.

**Table 2: Results for Experiment 1**

| Property | PRISM | COUNTER | $\Delta$ |
|---|---|---|---|
| R1 | 0.75299 | 0.753 | 0.00001 |
| R2 | 0.01595 | 0.016 | 0.00005 |
| R3 | 0.64400 | 0.630 | 0.01400 |

**Table 3: Results for Experiment 2**

| Property | PRISM | COUNTER | $\Delta$ |
|---|---|---|---|
| R1 | 0.92899 | 0.929 | 0.00001 |
| R2 | 0.01500 | 0.031 | 0.01600 |
| R3 | 0.59500 | 0.821 | 0.22600 |

Table 3 shows the analysis results for this experiment. Here we can see that $\Delta$ has increased for properties R2 and R3, specially the latter, when compared to the results obtained in the first experiment. In addition, according to this experiment, PRISM indicates that property R2 is valid, while R3 is violated. However, the Counter indicates that both these results are false, and in fact R2 is violated, while R3 is valid. Therefore, this experiment has generated both a false positive and a false negative.

## 2.2 Using a Behaviour Model with Improved Accuracy

**Experiment 3.** In this experiment we used the same EPBM used in Experiment 1, but a different BM. The BM

**Table 4: Results for Experiment 3**

| Property | PRISM | COUNTER | Δ |
|---|---|---|---|
| R1 | 0.75299 | 0.753 | 0.00001 |
| R2 | 0.01599 | 0.016 | 0.00001 |
| R3 | 0.62999 | 0.630 | 0.00001 |

**Table 5: Results for Experiment 4**

| Property | PRISM | COUNTER | Δ |
|---|---|---|---|
| R1 | 0.92899 | 0.929 | 0.00001 |
| R2 | 0.02500 | 0.031 | 0.00600 |
| R3 | 0.82099 | 0.821 | 0.00001 |

used here is a behaviourly equivalent model of the one used in the previous experiments, but structurally different, with the transition cycle representing the user choosing between sending vital parameters, pressing the panic button and stopping the system being unfolded twice.

The analysis results for this third experiment are shown in Table 4. Compared to the results of Experiment 1, which used the same EPBM, we can see that in this experiment there has been a significant improvement in the accuracy of properties R2 and R3, with all three properties presenting a small Δ.

**Experiment 4.** Finally, in this last experiment we used the same BM of Experiment 3 and the EPBM of Experiment 2. We ran the process again and obtained the results shown in Table 5. Similarly to what happened in Experiment 3, in this fourth experiment the probabilities computed for all three properties were also more accurate when compared to the results of Experiment 2, which used the same EPBM. In addition, the problem of having false predictions did not occur.

## 2.3 Discussion

The above experiments show that if we use a behaviour model that properly represents the probabilistic choices made by the environment, this model is accurate and, consequently, we can expect more accurate predictions. This was observed in Experiments 1, 3 and 4. On the other hand, using a behaviour model that does not take into account all the different probabilistic choices made by the environment may result in inaccurate predictions, as it happened for Experiment 2.

Therefore, in order to obtain accurate predictions, it is necessary that the system behaviour model has a representative structure. By representative we mean that the structure of the behaviour model has to accurately reflect and be consistent with the probabilistic choices made by the environment.

# 3. FORMAL BACKGROUND

In this section, we formalise the notions of structure, behaviour and probabilistic behaviour model used in the previous section and that will be referred to throughout the paper.

To describe system behaviour, we use the notion of Labelled Transition Systems [13], as formalised below:

[**Definition 1. LTS**] A labelled transition system (LTS) is a structure P=($S$, $A$, $\Gamma$, $q$) where:

– S is a finite set of states;
– $A = \alpha(P)$ is a set of labels that denotes the communicating alphabet of P;
– $\Gamma \subseteq (S \times A \times S)$ defines the labelled transitions between states;
– $q \in$ S is the initial state;

If $s,t \in$ S and $a \in$ A, we write $s \xrightarrow{a} t$ if $(s,a,t) \in \Gamma$.

Equivalence relations provide a semantic framework for constructing and comparing the behaviour represented by LTSs. They can also be used when reducing the state space of LTSs to simplify model analysis. We use the bisimulation relation to identify LTSs with the same branching structure.

[**Definition 2. Bisimulation**] A binary relation $\Re$ on the states of an LTS P=($S$, $A$, $\Gamma$, $q$) is a bisimulation if whenever $s \Re t$ and $a \in A$ then the following holds:

– if $s \xrightarrow{a} s'$, there exists $t'$ such that $t \xrightarrow{a} t'$ and $s' \Re t'$;
– if $t \xrightarrow{a} t'$, there exists $s'$ such that $s \xrightarrow{a} s'$ and $s' \Re t'$;

Two states $s$ and $t$ are said to be bisimular, denoted by $s \approx t$, in case $(s, t)$ is contained in some bisimulation $\Re$. Given two LTSs P and Q, we write P ≈ Q whenever $\alpha$(P)=$\alpha$(Q) and there is a bisimulation between the states of P and states of Q.

Let $\Re$(s) be the set of all the elements $t \in$ S such that $(s, t) \in \Re$. Hence, we write $\Re^{-1}$(t) to describe the set of all elements $s \in S$ such that $(s, t) \in \Re$.

One of the strongest equivalence relations is isomorphism. Two LTSs are isomorphic if their structure is exactly the same. To compare, trace equivalence preserves a minimal amount of structure: only the order in which actions can occur is preserved. Bisimilarity, on the other hand, also preserves the branching structure.

[**Definition 3. Isomorphism**] Two LTSs P = (S, A, Γ, q) and Q = (S', A', Γ', q') are isomorphic if, and only if, there exists a bijection f : S → S' such that f(q) = q' and $s \xrightarrow{a} s'$ iff f(s) → f(s').

Effectively, this means that isomorphic LTSs are only allowed to differ in their labelling of states. Given two LTSs P and Q, we write P is *structurally equivalent* to Q whenever $\alpha$(P)=$\alpha$(Q) and there is a isomorphism between P and Q, and P is *structurally different* from Q otherwise.

Probabilistic LTSs extend the ordinary LTS formalism by augmenting all transitions of the original LTS with a probabilistic choice for the possible target states, rather than a unique target state as it is the case in LTSs. That is, in the probabilistic setting, the transitions are of the form $s \xrightarrow{a} \mu$ where $s$ is the starting state, $a$ an action label and $\mu$ a transition probability function on the state space which specifies the probabilities $\mu(t)$, $0 \leq p \leq 1$, for any possible successor state $t$.

[**Definition 4. Probabilistic LTS**] A probabilistic LTS (PLTS) is a structure M = ($S$, $A$, $\Gamma$, $q$, $\mu$), where:

– ($S$, $A$, $\Gamma$, $q$) is an LTS;
– $\mu$: $(S \times A \times S) \rightarrow$ [0,1] is the transition probability function which assigns a positive real number less or equal to 1

for each transition such that the sum of the probability of all transitions leaving the same state is 1.

We write $P_\mu$ for an LTS P annotated with the transition probability function $\mu$. If $s,t \in S$ and $a \in A$, we write $s \xrightarrow{a[p]} t$ if $(s,a,t) \in \Gamma$ and $a$ occurs with probability $0 \leq p \leq 1$.

As in the LTS setting, there are different notions of equivalence for probabilistic LTSs, such as probability trace and failure equivalence. The behavioural equivalence we shall consider in this paper is probabilistic bisimulation, defined as:

**[Definition 5. Probabilistic bisimulation]** An equivalence relation $\Re$ over the states of a PLTS $M=(S, A, \Gamma, q, \mu)$ is a probabilistic bisimulation whenever $s \Re t$, then the following holds:

– $\forall a \in A$, for all equivalence classes C of $\Re$, $s \xrightarrow{a[p]} C \Leftrightarrow t \xrightarrow{a[p]} C$.

Two states $s$ and $t$ are said to be probabilistically bisimular, denoted by $s \equiv t$, in case $(s, t)$ is contained in some probabilistic bisimulation. Given two PLTSs P and Q, we write $P \equiv Q$ whenever $\alpha(P)=\alpha(Q)$ and there is a probabilistic bisimulation between the states of P and the states of Q.

Based on the previous definitions, we can formalise the problem that we are addressing as:

**Remark 1**. Let P and Q be two LTSs such that $P \approx Q$ and P is structurally different from Q. Let $\mu$ to be a transition probability function over the states of P. Then $\exists$ a transition probability function $\theta$, $\theta \neq \mu$, such that $Q_\theta \equiv P_\mu$.

Remark 1 is a typical assumption that other researchers take when probabilistically annotating an existing non probabilistic behaviour model and that may lead to inaccurate property prediction. We will show that this remark is false. To do so, we can take as an example the models $P_1$ and $P_2$ of Figure 3 representing the behaviour of a read-only variable. Suppose we apply to $P_1$ a transition function probability $\mu=\{(wait,0.1), (open,0.9), (read,0.3), (close,0.7)\}$. By hypothesis, we cannot apply the same transition probability function to $P_2$, then even if we repeat the probabilities of $\mu$ on the respective transitions of $P_2$, transitions *read* and *close* from state 2 would have to have different probabilities from the transitions with the same labels from state 1, preventing the probabilistic model of $P_2$ from being probabilistically bisimilar to $P_{1\mu}$. Therefore, whatever $\theta$ we choose, we will never be able to produce probabilistically bisimilar models.

# 4. STATE REFINEMENT

As described in Section 2, there can be benefit to be gained from refining behaviour models whose structure is not consistent with the probabilistic model of the environment. In this section, we describe the concept of state refinement, which has a key impact on the property prediction accuracy. We also demonstrate that by state refining a model we will always improve the model accuracy or, at least, produce a model as accurate as the previous one.

## 4.1 Concepts

Refinement is the process by which abstract specifications are transformed into more realistic designs or concrete implementations. This concept is frequently used, for example, in model-driven software development, where the process of development starts with an abstract model, which is refined in later design phases. The main usefulness of refinement is that the complex task of implementing a system that satisfies a given specification is made easier by gradually refining the abstract specification until finally a (concrete) implementation is obtained [12].

We define *state refinement* as the process by which a behaviour model is transformed into a bisimilar model by expanding a particular state, i.e., by introducing in the new model a copy of that state and its transitions such that we obtain two bisimilar states. The effect of it is a partition on the state set of the refined model such that each partition is abstracted by one state in the abstract model. Consequently, the refined and the abstract models are not isomorphic, since the former has a different structure from the latter. The aim of state refinement is to find a behaviourally equivalent model that improves the structure of the model being refined, making it more realistic with respect to the real system behaviour. We refine the states of a model based on substitutability, i.e., the replacement of the abstract model by its refinement cannot be observed. Formally, we define this process as:

**[Definition 6: State refinement.]** Given two LTSs $P_1$ and $P_2$, we write $P_1$ is state refined by $P_2$, denoted by $P_1 \sqsubseteq P_2$, if $\exists$ bisimulation $\Re$ between $P_1$ and $P_2$ such that if $s \Re t$, then $|\Re(s)| > 1$.

In section 2, the teleAssistance behaviour model used in Experiments 3 and 4 is a state refinement of the original model used in Experiments 1 and 2. Note that our concept of state refinement does not take into account the probabilities annotated on each model. In this way, our work differs from others in the probabilistic refinement field, such as [10].

Another important concept is the notion of abstract state, defined as:

**[Definition 7. Abstract state]** Given two LTSs $P_1=(S, A, \Gamma, q)$ and $P_2=(S', A', \Gamma', q')$, $P_1 \sqsubseteq P_2$, we say that a state $s$ of $P_1$ is an *abstract state* if $\exists t \in S'$, $s \Re t$, such that $|\Re(s)|>1$.

To illustrate this concept, consider the models shown in Figure 3. We have that $P_1 \sqsubseteq P_2$ and $P_2 \sqsubseteq P_3$. State 1 of model $P_1$ is an abstract state, since it is abstracting states 1 and 2 of $P_2$. We say that a state refinement is *valid* if it refines an abstract state. Hence, $P_2$ is a valid refinement of $P_1$. On the other hand, $P_3$ is an *invalid* state refinement of $P_2$, since it refines state 1 of $P_2$ and it is not an abstract state, given that $|\Re(\text{state 1 of } P_2)|=1$. In the rest of the paper, whenever we refer to a state refinement, we assume that it is a valid refinement.

Although the state refinement process is carried out in the non-probabilistic model, it is necessary to annotate probabilities on the refined model in order to verify whether it is accurate or not. Thus, we introduce the concept of *Annotator*, which is responsible for inserting the probabilities into the respective transitions of a behaviour model, yielding its probabilistic counterpart. Formally, we define the *Annotator* as:

**[Definition 8: Annotator.]** Given an LTS P and a transition probability function $\mu$ for P, we define *Annotator(P,$\mu$)* an algorithm that receives P and $\mu$ and produces a PLTS

$P_\mu$ by annotating the probabilities from $\mu$ on the respective transitions of P.

We assume that whenever a model is state refined, its respective transition probability function is also refined in a sense that it will always provide the best transition probability function for each state of the new model, i.e., the probabilities are as accurate as possible for those transitions. Although this is a strong assumption, we argue that this is what happens in the real world, since the probabilistic information that was misrepresented in the first model can be more properly represented in the new model.

Given a property $\varphi$, we denote by $\Delta_{P\mu}$ the absolute difference between the result of the probabilistic verification of $\varphi$ against the annotated model $P_\mu$ and the result of the probabilistic verification of $\varphi$ against the real probabilistic system behaviour model.

## 4.2 Properties of State Refinement

Here we present the state refinement properties that guarantee that accuracy is always preserved or increased when refining a behaviour model and that after successive refinements we will find a model that is accurate and for which no more refinements are needed.

***Transitivity***: This property states that *if* $P_1 \sqsubseteq P_2$*, and* $P_2 \sqsubseteq P_3$*, then* $P_1 \sqsubseteq P_3$. This is true because if one state of $P_1$ abstracs two states of $P_2$, and one of these states of $P_2$ abstracts two more states of $P_3$, then it means that the initial abstract state of $P_1$ also abstracts those three states of $P_3$, thus by definition of state refinement, we have that $P_1 \sqsubseteq P_3$. For example, in Figur 3, model $P_1$ is refined by model $P_2$, that is refined by model $P_4$. By this property, we have that model $P_1$ is also refined by model $P_4$.

***Preservation or improvement of the accuracy***: This property states that *by state refining a model we always improve the accuracy or we do not obtain worse predictions.* To prove this property, consider the following theorem:

**[Theorem 1.]** Let $P_1$ and $P_2$ be two LTSs, $\mu_1$ and $\mu_2$ be two transition probability functions and $\varphi$ be a property. If $P_1 \sqsubseteq P_2$, Annotator $(P_1,\mu_1) = P_{1\mu1}$ and Annotator $(P_2,\mu_2) = P_{2\mu2}$, then $\Delta_{P1\mu1}(\varphi) \geq \Delta_{P2\mu2}(\varphi)$

In the worst case, we have $\mu_1 = \mu_2$, i.e., *Annotator* will repeat the same transition probability function for each transition leaving bisimilar states in both models. Then we obtain probabilistically bisimilar models, which means that we have $\Delta_{P1\mu1}(\varphi) = \Delta_{P2\mu2}(\varphi)$. However, as $P_2$ is a refinement of $P_1$, then by the assumption that *Annotator* always generates the best probabilistic model, we have that $\mu_2$ is more accurate than $\mu_1$. Hence, *Annotator* generates a probabilistic model with more accurate probabilities, which implies in $\Delta_{P1\mu1}(\varphi) > \Delta_{P2\mu2}(\varphi)$. Therefore, $\Delta_{P1\mu1}(\varphi) \geq \Delta_{P2\mu2}(\varphi)$.

***Convergence***: This property states that *there is at least one sequence of refinement steps that always generates an accurate model and that, at a certain point, no more refinement will be necessary.* One model can have several refinements, since it can have many abstract states. Furthermore, the same abstract set can be refined in different ways. Then, by the previous property, at least one of the refined models will be more accurate than the abstract one. If we choose the most accurate refined model and refine it, then it will generate a more accurate model yet. By repeating the process successively, we will always obtain a better model until we reach a model that produces results as accurate as the real probabilistic system behaviour model. At that moment, no more refinement is necessary and an accurate model is obtained.

Therefore, there is a real gain in accuracy when applying state refinement to a model. Then, we have to work out how to choose the best refinement at each refinement step such that we obtain an accurate model at the end. The next section discusses this issue.

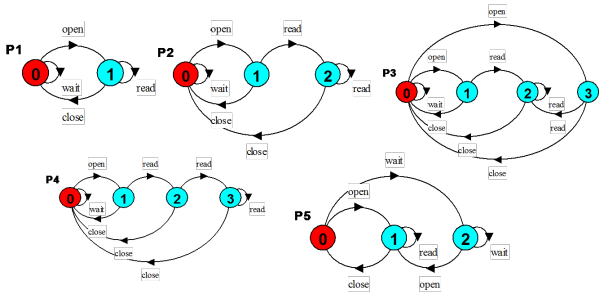


**Figure 3: Examples of state refinement.**

# 5. OBTAINING AN ACCURATE MODEL

In the previous section, we introduced the concept of abstract state. An inaccurate model has at least one abstract state. In addition, by transitivity, we have that the same abstract state, refined many times, will generate many refined models. Thus, we have that an abstract state abstracts a *set of states* with respect to a further refined model. As a model can have different abstract states, we have the notion of abstract state set, defined as:

**[Definition 9. Abstract state set]** Given two LTSs $P_1$=(*S, A, Γ, q*) and $P_2$=(S', A', Γ', q'), $P_1 \sqsubseteq P_2$, the Abstract state set of $P_1$ with respect to $P_2$, denoted by Abs($P_1$,$P_2$), is the set of pairs $(s,t)$, where $s \in$ S, $t \in$ S', such that $s$ is an abstract state and $s \Re t$.

For example, consider the models shown in Figure 3. We have that $P_1 \sqsubseteq P_2$, $P_1 \sqsubseteq P_4$ and $P_1 \sqsubseteq P_5$. We also have that Abs($P_1$,$P_2$)= {(1,1),(1,2)}, Abs($P_1$,$P_4$)= {(1,1),(1,2),(1,3)}, and Abs($P_1$,$P_5$)= {(0,0),(0,2)}.

Now assume that model $P_4$ of Figure 3 is the real system behaviour model and model $P_1$ is the original (provided) behaviour model. Note that Abs($P_1$,$P_2$) $\subset$ Abs($P_1$,$P_4$), while Abs($P_1$,$P_5$) is not a subset of Abs($P_1$,$P_4$). This shows that $P_2$ decreased the number of abstract states of $P_1$ with respect to $P_4$. In addition, we can see that $P_2$ has a structure *closer* to $P_4$ than $P_5$ does. Therefore, there is a relation between the abstract state set of a model and the *distance* of that model to the real system model: the more abstract states a model has with respect to the real model, the larger the distance to that model is, and the less accurate the model is.

Given $P_1$, $P_2$, and $P_3$ three LTSs, if $P_1 \sqsubseteq P_2$, $P_2 \sqsubseteq P_3$, we write that $P_2$ *is closer to* $P_3$ *than* $P_1$ *is* if Abs($P_1$,$P_2$) $\subset$ Abs($P_1$,$P_3$). Hence, among the possible refinements of a model, the more accurate is the one for which the abstract state set with respect to the real model is a subset of the

abstract state set of the abstract model with respect to the real model.

Although useful, this concept of distance assumes that the real system behaviour model is provided. In the real world, it is unlikely that that model will be known beforehand, since we may not have any information about the real probabilistic choices of the environment. However, we can show that among several refined models, the closest one to the real model is the one that decreases $\Delta$ with respect to the given set of properties.

**[Theorem 2.]** Let $P_1$, $P_2$, and $P_3$ be three LTSs such that $P_1 \sqsubseteq P_2$, $P_2 \sqsubseteq P_3$. Let $\varphi$ be a property and $\mu$ be a transition probability function such that Annotator($P_1$,$\mu$)=$P_{1\mu}$ and Annotator($P_2$,$\mu$)=$P_{2\mu}$. If Abs($P_1$,$P_2$) $\subset$ Abs($P_1$,$P_3$) $\Rightarrow$ $\Delta_{P1\mu}(\varphi) > \Delta_{P2\mu}(\varphi)$.

As $P_1 \sqsubseteq P_2$, by Theorem 2 we have that $\Delta_{P1\mu}(\varphi) \geq \Delta_{P2\mu}(\varphi)$. However, as $P_2$ is closer to $P_3$ than $P_1$ is, $P_2$ has less abstract states than $P_1$. This way, Annotator($P_{2,\mu}$) should produce a better model than Annotator($P_{1,\mu}$), which implies that $\Delta$ between $P_1$ and $P_2$ cannot be equal. Therefore, $\Delta_{P1\mu}(\varphi) > \Delta_{P2\mu}(\varphi)$.

Hence, we have shown that choosing the best refinement means selecting the model that has the smallest $\Delta$. Finally, we have to show that after successive refinements we will obtain an accurate model.

**[Theorem 3.]** Let $P_1$, $P_2$ be two LTSs and let $\mu$ be a transition probability function. If $P_1 \approx P_2$, $P_1 \neq P_2$ and Abs($P_1$,$P_2$) = $\emptyset$, then Annotator($P_1$,$\mu$) $\cong$ Annotator($P_2$,$\mu$).

The fact that Abs($P_1$,$P_2$) = $\emptyset$ means that $P_1$ is not abstracting any states of $P_2$. In this way, if we apply the same transition probability function to both models we will obtain that the bisimilar states will have their transitions annotated with the same probabilities. Hence, $P_{1\mu} \cong P_{2\mu}$.

Therefore, when state refining an abstract model, if we always choose the refined model that produces the smallest $\Delta$ for the given set of properties, and successively refine it, we will obtain an accurate model.

# 6. THE FRAMEWORK

Based on the theory of state refinement presented in the previous section, we propose a framework to iteratively refine a provided behaviour model until we obtain an accurate model. The framework, shown in Figure 4, is a generalisation of the behaviour analysis process used to verify the accuracy of the models in Section 2 since, in practice, we may not know how the environment behaves probabilistically. In addition, it may be the case that the system under analysis does not exist yet. Hence, we replace the traces generated by the simulation by a generic source of probability, which also encloses, for instance, estimates and operational profiles.

Broadly, our framework is divided into three phases: Phase 1 leverages a system behaviour model and a source of probability to obtain a probabilistic behaviour model; Phase 2 takes the probabilistic model of Phase 1 and verifies its accuracy with respect to a group of properties; finally, if the model is not accurate, it is inputted into Phase 3, where it will be state refined, producing new models. These refined models then feed Phase 1 and the whole process is carried out again for each one, when the most accurate model among them is determined. If it is still inaccurate, then the model is refined. The process terminates when an accurate model is obtained. The verification is based on a *benchmark* of properties, for which a true value is provided by an *oracle*, and it is carried out through a comparison between the result obtained from a probabilistic verification of each property and the result from the oracle. A model is accurate if the property prediction generates values similar to those of the oracle. More details about each phase are provided in the following sections.

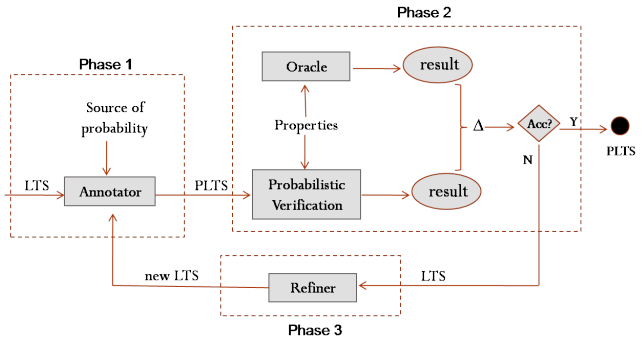


**Figure 4: Framework for state refinement.**

## 6.1 Phase 1: Obtaining the Probabilistic Behaviour Model

In this phase, we obtain a probabilistic behaviour model for the system under analysis by annotating the occurence probability of each transition on the provided system behaviour model. We assume that the provided model is correct in that it does model the actual system behaviour. The probabilities can come from different sources, such as estimates made by an expert on the system, traces generated by monitoring the system execution or an operational profile, as proposed by [18]. We can see the source as a transition probability function that assigns a number less or equal to 1 for each transition of the model.

We assume that the source of probability is complete with respect to the model, i.e., it provides probability for each transition of the model. However, if it is not complete, some heuristics could be applied, such as annotating the transitions leaving the same state equiprobably. It is out of our scope to propose a method to generate the source of probability.

The Annotator is responsible for inserting the probabilities in the behaviour model, yelding a PLTS.

## 6.2 Phase 2: The Model Accuracy Verification Process

After having generated the probabilistic behaviour model, the accuracy of the model is verified in Phase 2. In this section we describe the elements involved in the verification process and explain how we determine whether a model is accurate or not.

### 6.2.1 *Benchmark of properties*

The accuracy of a model is determined with respect to a group of properties, called the *benchmark*, which is provided by the framework user.

In our framework, we use a particular kind of properties to guide the state refinement process: properties that include

actions representing choices made by the environment. For instance, all properties used in Section 2 are in that set, since they involve the probability of the user choosing between sending the vital parameters, pressing a panic alarm button and stopping the system. Suppose, for instance, another TA property described by [11] that says: "assuming that alarms generated by *pButtonMsg* have low priority while alarms generated by *analyzeData* have high priority, it is required that the probability that a high priority alarm fails (i.e., it is not notified to the FAS) is less then 0.012". Looking at the model depicted by Figure 2(a), we see that this property regards only the transitions *analyseData* and *sendAlarm*, which are internal to the system. In this way, whatever the user does, it does not affect the result of the property. As this kind of property is never affected by our state refinement process, it cannot be used in the benchmark to guide the model refinement process.

Each property is verified using a type of probabilistic verification, such as numerical analysis or probabilistic model checking. Given a property $\varphi$ and a PLTS $P_\mu$, we write *prob($\varphi$,$P_\mu$)* for the probability of the property $\varphi$ holding on $P_\mu$ obtained from a probabilistic verification process.

### 6.2.2 Oracle

In order to verify whether the probabilistic model is accurate, the framework assumes that an *oracle* provides the true value for each property of the benchmark. An oracle can be, for instance, an expert on the system, such as the developer or the user, who provides values for each property based on their knowledge, or a set of traces of the system, obtained from logging the system execution. Given a property $\varphi$, a PLTS $P_\mu$, and an oracle O, we write *prob($\varphi$,$P_\mu$,O)* for the probability of the property $\varphi$ holding on P provided by the oracle O.

There is a concern regarding the reliability of the oracle. For instance, an oracle based on estimates may be less reliable than an oracle based on the system traces. Nonetheless, even using traces, a small sample set may be less reliable than a larger one. We address this problem by rating the oracle with the parameter $\delta$ that indicates the imprecision of the oracle. If $\delta$ is high, so the oracle is more likely to provide an imprecise value. This parameter is provided by the framework user and works as an acceptable variation on the measure of the accuracy of the model.

### 6.2.3 Accuracy verification

The accuracy verification is carried out by comparing the result obtained from the probabilistic verification to the result provided by the oracle, regarding the same property. We define the measure $\Delta$ as:

[**Definition 10. Difference between predictions**] Given a PLTS P, an Oracle O, and a property $\varphi$, the distance of the predictions ($\Delta$) is calculated as: $\Delta_{P\mu}(\varphi) = |prob(\varphi, P_\mu, O) - prob(\varphi,P_\mu)|$

The accuracy is a subjective measure. For instance, some users might say that if the difference between the predictions is less than 0.01, then the result is accurate. On the other hand, there may be users for whom an accurate result should be one whose $\Delta$ is less than 0.0001. We call the *accuracy threshold* $\epsilon$ the maximal acceptable $\Delta$. The less the thresold is, the more precise the result will be.

Now, we can define our measure of model accuracy as:

[**Definition 11. Model accuracy**] A PLTS $P_\mu$ is *accurate* with respect to a property $\varphi$ if $\epsilon - \delta \leq \Delta_{P\mu}(\varphi) \leq \epsilon + \delta$.

Note that we take into account the oracle reliability when stating the model accuracy. In other words, by considering this parameter we are extending the accuracy threshold, so it can vary depending on the precision provided by the oracle. The more reliable the oracle is, the less the interference from $\delta$ on $\epsilon$ there will be.

Finally, the model is outputted by the framework if it is accurate. Otherwise, it is inputted into the next phase to be refined.

## 6.3 Phase 3: Refinement

In this phase, the inaccurate model is state refined into a group of new models. Then, each model goes through the whole process in order to find which one produces better predictions. Once we determine that model, we check if it is accurate. If that model is still inaccurate, then it is refined again, and the same procedure is performed for the new refinements.

The state refinement carried out in this phase consists of choosing a specific abstract state then expanding it. We consider a state an abstract one if it has more than one outgoing transition and if it can be reached by following all possible paths started from that same state. Once we identify an abstract state, we generate a group of refined models by expanding it. There are different approaches to expand a state, but all of them share the process of copying the state and all further states that are in the branch started by the abstract state, as well as their respective transitions. The difference affects how this new branch will be linked to the rest of the model: if by changing all incoming transitions of the abstract state to this new branch, or if by linking only some of those transitions.

# 7. CASE STUDY

In this section, we describe an application of our framework and discuss the results we obtained. Although we validate our framework through analysing a simulation based on a real system, the framework is general and can be used in the case where the system does not yet exist, such as for predicting system reliability. In this setting, it could be used as an extension of the approach described in [7].

We focus on the TA system, the same used in section 2, and specifically on checking property R2. The goal of the evaluation is to assess the gain in accuracy that the framework introduces, taking into account the number of refinement steps necessary and the size of the final model. We achieve this goal by instantiating the framework like this: we used a simulation of the TA, which is consistent with the provided original system behaviour model, and generated a set of traces randomly. The traces are our source of probabilities and are used as input to Annotator and Counter, the same algorithms used in the experiments of section 2, implemented here as Java programs. We assume Counter as being our Oracle that will provide the accurate result for the target property. We used PRISM to model-check probabilistically property R2 against each refined model. Our model refiner was also implemented as a Java program that produces as output the possible refined models for a given input model. Due to space limitations, in this section we only compare the results of three refinements for each model.

According to our Oracle, R2 occurs with probability 0.038 (hence the property is violated). We assume the value 0.0001 as both our accuracy threshold and the imprecision of the Oracle, which relies on 1000 traces as the data set. Thus, we consider that we have an accurate prediction if the result of the probabilistic model check of R2 is between 0.0378 and 0.0382. Figure 5 depicts the refinement steps taken during the application of the framework. Each node of that refinement graph represents a behaviour model, where the root is the provided behaviour model. The probability beside each node is the result generated by PRISM by model checking R2 for that model. Each level of the graph represents a refinement step taken during the refinement process. For each model, three refinements were generated and their probabilities compared in order to determine which one was the more accurate.

From Figure 5, we can see that although the initial behaviour model produces neither a false negative nor a false positive, it generates a very imprecise result if compared to the accurate model (the right most node of the last level), for which the R2 probability is more than double the R2 probability of the original model. However, by successively refining that initial model, we always preserve the previous probability or increase it, as we can see at each applied refinement step, until we reach an accurate model for which no more refinement is necessary. In this case study, three refinement steps were needed to obtain an accurate model.

The drawback of our state refinement technique is the increase of the model state set. In this case study, the final model is more than eight times larger (w.r.t. the number of states) than the initial one. This problem may lead to state space explosion, since the refined model might have to grow exponentially in order to obtain an accurate model, depending on how the environment uses the system. On the other hand, the benefit of having an accurate model gives us confidence to reason about future properties and new requirements.

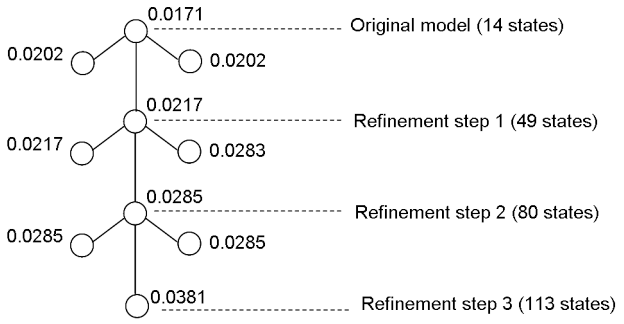


**Figure 5: Refinement steps and accuracy results from the application of the framework.**

# 8. RELATED WORK

Quantitative behaviour analysis has been the focus of intensive research in the last decades, and applied to address different aspects of software behaviour within different problem domains [22, 10, 21, 19, 7, 11, 2]. In this section we discuss some of the existing behaviour analysis approaches and how they are related to our work.

In [19], the authors describe an approach for predicting software reliability in which generative probabilistic behaviour models are synthesised for each system component from scenarios annotated with scenario transition probabilities and component reliability information. The resulting probabilistic models are then composed in parallel and used to predict the overall system reliability. In that work, scenario transition probabilities are assumed to come from an operational profile of the system, while component reliabilities are simply estimated. A framework specifically aimed at predicting component reliability at the architectural level is described in [7], where Hidden Discrete Time Markov chains are used to model normal and faulty component behaviour. State and transition probabilities in the Markov chains can be determined based on a combination of multiple information sources, including operational profiles, architectural behaviour models, estimates by domain experts, requirements documents, and simulation, thus avoiding their individual limitations and inaccuracies. Component reliability is computed by solving the Markov chain model using standard numerical techniques. Finally, a more recent behaviour analysis approach is described in [11], where the authors also use Discrete Time Markov chains to model and quantify non-functional properties. However, in that work behaviour models are continuously updated with new transition probabilities obtained from the running system, which the authors claim achieves increasingly better prediction accuracy.

All the above approaches share our goal of using probabilistic behaviour models to quantitatively predict system (in the case of [19] and [11]) or component (in the case of [7]) level properties. In addition, they all share our concern that having accurate models is key to improve prediction accuracy. However, none of those approaches take into account the fact that the *structure* of the target behaviour model (and not only its state and transition probabilities) may have a strong influence in the analysis results. As we have shown in the paper, annotating an existing behaviour model with probabilistic information, even when using reliable information sources, can lead to inaccurate predictions if the model does not properly represent the choices made by the environment.

Our work is more closely related to the approach described in [10], where the authors propose a method to model-check quantitative properties using behaviour models specified as Markov Decision Processes. In particular, we share the same approach of starting from an initial model and gradually refining it until we obtain a satisfactory (i.e., accurate) prediction or no further refinement is possible. However, in that work the authors apply model refinement to determine whether the probability of reaching a particular final condition from any reachable state satisfying a given initial condition is smaller or greater than a given value, while our purpose is to find a structure that better represents the probabilistic choices made by the environment with respect to a given set of properties.

# 9. CONCLUSIONS AND FUTURE WORK

In order to obtain accurate results in probabilistic modelling, we argue that it is important that both the probabilities be accurate and the model itself accurately reflects reality. We have shown an example which illustrates this, indicated its consequences, and proposed a framework to obtain accurate models through successive state refinements.

Our contribution is twofold. Firstly, we presented the

state refinement technique to mitigate inaccuracy prediction, and showed that it is guaranteed to preserve or increase the accuracy of the initial annotated model. Secondly, we provided a working framework for iteratively refining a model until an accurate model is obtained. We also introduced the concept of benchmarking of properties to guide our state refinement process.

Currently, we are investigating how to avoid the state space explosion in case of a large model. In addition, we are conducting more case studies in order to validate and improve the framework. Finally, we intend to make the solution presented in this paper available through an integrated tool based on LTSA[17].

## 10. ACKNOWLEDGEMENTS

This research is sponsored by CAPES (Brazil) under grant no. BEX 4257-05/7.

## 11. REFERENCES

[1] R. Alur, L. J. Jagadeesan, J. J. Kott, and J. E. Von Olnhausen. Model-checking of real-time systems: a telecommunications application: experience report. In *ICSE '97: Proceedings of the 19th International Conference on Software Engineering*, pages 514–524, Boston, USA, 1997. ACM.

[2] C. R. Anderson, P. Domingos, and D. S. Weld. Relational markov models and their application to adaptive web navigation. In *KDD '02: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 143–152, Edmonton, Canada, 2002. ACM.

[3] C. Baier, F. Ciesinski, and M. Groesser. Quantitative analysis of distributed randomized protocols. In *FMICS '05: Proceedings of the 10th international workshop on Formal methods for industrial critical systems*, pages 2–7, Lisbon, Portugal, 2005. ACM.

[4] C. Baier and M. Kwiatkowska. Model checking for a probabilistic branching time logic with fairness. *Distrib. Comput.*, 11(3):125–155, 1998.

[5] A. Bianco and L. de Alfaro. Model checking of probabilistic and nondeterministic systems. *FSTTCS: Foundations of Software Technology and Theoretical Computer Science*, 15:499–513, 1995.

[6] W. Chan, R. J. Anderson, P. Beame, D. Notkin, D. H. Jones, and W. E. Warner. Optimizing symbolic model checking for statecharts. *IEEE Trans. Softw. Eng.*, 27(2):170–190, 2001.

[7] L. Cheung, R. Roshandel, N. Medvidovic, and L. Golubchik. Early prediction of software component reliability. In *ICSE '08: Proceedings of the 30th international conference on Software engineering*, pages 111–120, Leipzig, Germany, 2008. ACM.

[8] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. Program. Lang. Syst.*, 8(2):244–263, 1986.

[9] C. Courcoubetis and M. Yannakakis. The complexity of probabilistic verification. *J. ACM*, 42(4):857–907, 1995.

[10] P. R. D'Argenio, B. Jeannet, H. E. Jensen, and K. G. Larsen. Reachability analysis of probabilistic systems by successive refinements. In *PAPM-PROBMIV '01*, pages 39–56, Aachen, Germany, 2001. Springer-Verlag.

[11] I. Epifani, C. Ghezzi, R. Mirandola, and G. Tamburrelli. Model evolution by run-time adaptation. In *ICSE '09: Proceedings of the 31st International Conference on Software engineering*, Vancouver, Canada, 2009. ACM.

[12] R. Eshuis and M. M. Fokkinga. Comparing refinements for failure and bisimulation semantics. *Fundam. Inf.*, 52(4):297–321, 2002.

[13] R. M. Keller. Formal verification of parallel programs. *Commun. ACM*, 19(7):371–384, 1976.

[14] M. Kwiatkowska. Quantitative verification: models, techniques and tools. In *ESEC-FSE companion '07: The 6th Joint Meeting on European software engineering conference and the ACM SIGSOFT symposium on the foundations of software engineering*, pages 449–458, Dubrovnik, Croatia, 2007. ACM.

[15] M. Kwiatkowska, G. Norman, and D. Parker. Prism 2.0: A tool for probabilistic model checking. In *QEST '04: Proceedings of the The Quantitative Evaluation of Systems*, pages 322–323, Enschede, The Netherlands, 2004. IEEE Computer Society.

[16] M. Kwiatkowska, G. Norman, and D. Parker. Probabilistic model checking in practice: case studies with prism. *SIGMETRICS Perform. Eval. Rev.*, 32(4):16–21, 2005.

[17] J. Magee and J. Kramer. *Concurrency: State Models and Java Programs*. Wiley, 2006.

[18] J. D. Musa. Operational profiles in software-reliability engineering. *IEEE Softw.*, 10(2):14–32, 1993.

[19] G. Rodrigues, D. Rosemblum, and S. Uchitel. Using scenarios to predict the reliability of concurrent component-based software systems. In *FASE'05 / ETAPS 2005: 8th International Conference on Fundamental Approaches to Software Engineering*, pages 111–126, Edinburgh, Scotland, 2005.

[20] M. Y. Vardi. Automatic verification of probabilistic concurrent finite state programs. In *FOCS'85: Proceedings of the 26th Annual Symposium on Foundations of Computer Science*, pages 327–338, Baltimore, USA, 1985. IEEE Computer Society.

[21] J. A. Whittaker and M. G. Thomason. A markov chain model for statistical software testing. *IEEE Trans. Softw. Eng.*, 20(10):812–824, 1994.

[22] C. Wohlin and P. Runeson. Certification of software components. *IEEE Trans. Softw. Eng.*, 20(6):494–499, 1994.

[23] S.-H. Wu, S. A. Smolka, and E. W. Stark. Composition and behaviors of probabilistic i/o automata. *Theor. Comput. Sci.*, 176(1-2):1–38, 1997.