Model-Based Performance Prediction by Statistical Model Checking: An Industrial Case Study of Autonomous Transport Robots

Ryota Arai
Sony Corporation
Tokyo, Japan

Bernd-Holger Schlingloff
Humboldt Universitat zu Berlin, and
Fraunhofer FOKUS, Berlin

Abstract— **Performance is one of the most important software quality attributes. It heavily relies on system design and software architecture. Therefore, performance evaluation in the early design phases is important to avoid a time-consuming costly trial-and-error approach. Model-based performance prediction provides means of well-informed trade-off decisions. It answers whether the software system will meet its performance goals, and which of two or more competing design alternatives is the most suitable. This paper investigates the use of model checking for the analysis of performance requirements in industrial scale problems. We model an actual case study of autonomous transport robots in production plants, to experiment with stochastic model checking for the prediction of system performance. Via the case study, we show a modelling strategy, a way to measure different performance metrics, and the resulting trade-off decisions which can be made among design alternatives.**

I. Introduction

The characteristics and success of a software based system is determined not only by its functionality, but also by non-functional requirements. Important quality attributes comprise requirements on performance, reliability, usability, security, availability, and others [1]. Such non-functional requirements may be driven by business considerations such as costs and time-to-market. To create a successful system, not only the functional requirements, but also the quality attributes must be satisfied. However, often single software quality attributes can not be considered in isolation. Software quality attributes may depend on one another, or be in conflict with each other. Finding the right balance between competing software quality attributes often is the key point in the construction of a system. A survey [2] of industrial software development reports that usability, performance and reliability are the three most important quality aspects. While usability is a main quality goal for consumer systems, it is largely a function of the user interface. On the other hand, performance and reliability heavily rely on the system design and software architecture. In the software industry, performance investigations are often deferred until an implementation of the system has been build and measurements can be conducted. Performance problems detected at this stage may be so severe that they can require considerable changes in the design, for example, at the software architecture level. With such a development process, performance improvement often becomes a time-consuming costly trial-and-error approach.

Therefore, it is necessary to develop methods for analyzing and balancing non-functional requirements early in the development process. In a systematic engineering approach, performance should be predictable before the system is actually realized. The predictions should be based on scientific theories that enable a forecast, a kind of what-if analysis, already on the model level [3]. *Model checking* is a prominent approach to verify software design models. Safety standards like IEC 61508 recommend the use of formal verification, e.g., model checking, for achieving the highest safety integrity level. Therefore, large research efforts have gone into the development of model checking tools and techniques for safety-critical systems. However, model checking has not been widely used for the performance prediction of systems. An industrial survey [4] showed that the most common architecture review technique is experience-based reasoning (83 percent). The use of mathematical analysis and model checking is still not widely accepted.

In this paper, we investigate the use of model checking for the analysis of non-functional requirements in industrial scale problems. We model an actual case study of autonomous transport robots in production plants, to experiment with stochastic model checking for the prediction of system performance. Via the case study, we show a modelling strategy, a way to measure different performance metrics, and the resulting trade-off decisions which can be made among design alternatives. Our experiments show that model checking is useful to make well-informed decisions on various parameters of the design.

The rest of the paper is structured as follows: In section 2, we briefly review performance metrics and mathematical models. Section 3 recalls probabilistic model checking and how it provides means of performance evaluation. Our case study of autonomous transport robots is presented in section 4. We discuss the results and implications in section 5. Section 6 reviews related work. Finally, we conclude our paper and outline future work.

II. Performance Prediction

A. Metrics

Non-functional requirements are often specified ambiguously. For example, the statement *"The response time must be as fast as possible"* is difficult to verify. Therefore, the requirement should be refined to a precise quantitative form, e.g., *"The response time shall be less than 0.5 seconds in 98 percent of all user input actions"*. Performance is described by specific metrics. The most important ones are response time, throughput, and device utilization [5]. The response

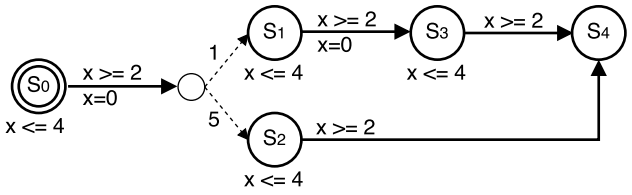

Fig. 1. A stochastic timed automata

time of a system is defined as the difference between the time at which an activity is completed and the time at which it was initiated. The throughput of a system refers to the rate at which a system handles requests; it is usually measured in tasks per time unit. The device utilization refers to the ratio of busy time of a resource by the total elapsed time of a measurement period.

B. Performance Prediction

The number of parameters of a system that may vary independently is called its *degrees of freedom*. Generally, a projected software has many degrees of freedom, and, thus, many design alternatives. In order to balance non-functional requirements, software engineers have to explore these alternatives and find optimal values for the parameters of the system. This design space exploration can be done using mathematical models and metrics. The model is solved using either analytical, numerical or simulation techniques. A quantitative prediction derived from such models enables well-informed trade-off decisions in the early design phases. It answers the question whether the software system will meet its performance goals, and which of two or more competing design alternatives is the most suitable.

C. Mathematical Modelling

Historically, the first performance evaluation methods were based on queueing theory. In this theory, a queueing model is constructed so that queue lengths and waiting time can be predicted. This method has been widely applied especially in telecommunications. Many systems, especially embedded systems, require models integrating both real-time constraints and probabilistic aspects. Stochastic timed automata (STA) [6] integrate both features into an extension of timed automata (TA). TA are labeled transition systems with clocks to measure time elapsed, guards to specify when an action is enabled, and urgency constraints to force actions to happen at some ultimate time instant. STA incorporate branching edges where weights can be added to give a distribution on discrete transitions and allow probabilistic transitions. Fig. 1 shows an example of STA and its distribution of the overall reachability time to S_4 is given in Fig. 2. The timing behavior is controlled by clock x. The states $S_{\{0,1,2,3\}}$ have the invariant defined $x \leq 4$. Each transition has the guard condition defined $x \geq 2$ and clock x is reset. Thus transitions occur when the clock value is $2 \leq x \leq 4$. In addition, the transition from S_0 follows the probabilistic choice ($\frac{1}{6}$ and $\frac{5}{6}$).

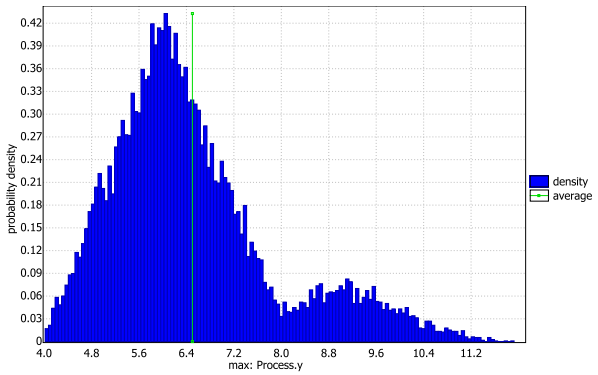

Fig. 2. A distribution of reachability time

III. Model Checking Approach

This section recalls the basic concept of probabilistic model checking and how it provides means of performance analysis. Afterwards, we show comparisons of numerical and statistical approaches in probabilistic model checking. Finally, we give a brief introduction to UPPAAL SMC which ii a tool for statistic model checking.

A. Probabilistic Model Checking and Performance Analysis

Model checking is an automated verification technique that explores all possible system states of a finite-state model to check whether a formal property holds for that model [7]. Typically, formal properties are expressed in temporal logic, e.g., LTL and CTL. Traditionally, model checking has been used to verify the functional correctness of systems, which is represented as safety, liveness and fairness properties. The extension of temporal logic with probabilistic property, such as, Continuous Stochastic Logic (CSL) [8], Probabilistic real time Computation Tree Logic (PCTL) [9] and Weighted Metric Temporal Logic (WMTL) [10], provides a powerful means to specify both path-based and traditional state-based performance measures on probabilistic models. The probabilistic model checking incorporates the performability-like evaluation techniques into model-checking tools. As described in [11], the largest advantage of model checking for performance analysis is that all algorithmic details, all detailed and non-trivial numerical computation steps are hidden to the user. Without any expert knowledge on numerical analysis techniques, performance evaluation is possible.

B. Statistical Model Checking

One of the major practical obstacles shared by model checking is the state space explosion problem. Industrial software systems often have massive states. Scaling model checking technique to large systems has been a challenge for decades and driven the development of various techniques to combat against the state space explosion problem.

Probabilistic model checking is classified into numerical and statistical approaches. The numerical approach can often provide a higher accuracy than the statistical approach, whose results are probabilistic in nature. However, the numerical approach is far more memory intensive. Therefore,

the state explosion problem is even more severe. On the other hand, the statistical approach uses far less memory and is time intensive. A survey [12] demonstrates that the complexity of both the numerical and the statistical approach is typically linear in the time-bound of the property, but that the statistical approach scales better with the size of the state space. Furthermore, the statistical approach requires considerably less memory than the numerical approach, allowing us to verify models far beyond the scope of numerical solution methods.

The core idea of statistical model checking (SMC) is to monitor some simulations of the system, and then use results from the statistics area (including sequential hypothesis testing or Monte Carlo simulation) to decide whether the system satisfies the property with some degree of confidence. SMC can be seen as a trade-off between testing and formal verification.

C. UPPAAL SMC Model Checker

UPPAAL SMC [13] is an extension of UPPAAL for statistical model checking. The model checker verifies quantitative properties by estimating the probabilities and probability distributions over time with given confidence levels. The tool consists of a graphical modeling editor, a simulator, and a verifier. The modeling formalism of UPPAAL SMC is based on a stochastic timed automata (STA). The component STAs communicate via broadcast channels and shared variables to generate Networks of Stochastic Timed Automata (NSTA). In the verifier, a formal property is specified by Weighted Metric Temporal Logic (WMTL) and it is mainly classified into three types:

Probability estimation: The probability $P_M(\Diamond_{x\leq C} ap)$ for a given NSTA M can be estimated by the query: "Pr[bound] (ap)", where ap is a conjunction of predicates over the state of a NSTA and bound defines how to bound the runs. The bound is defined either by time, by cost or by number of discrete steps.

Hypothesis testing: Whether the probability $P_M(\Diamond_{x\leq C} ap)$ for a given NSTA M is greater or equal to a certain threshold $p \in [0, 1]$ can be estimated by the query: "Pr[bound] (ψ) >= p". The formula ψ is either $<>q$ or $[]q$ where q is a state predicate.

Probability comparison: Whether the probability $P_M(\Diamond_{x\leq C} ap_1)$ is greater than $P_M(\Diamond_{x\leq C} ap_2)$ can be estimated by the query: "Pr[bound1](ψ_1) >= Pr[bound2](ψ_2)".

IV. Case Study

We investigate the use of model checking for the analysis of performance requirements in industrial scale problems. In this section, we give a general idea of autonomous transport robots and its typical usage scenario. Then we address a performance modeling and a way to measure performance metrics.

A. Autonomous Transport Robots

A collaborative embedded systems (CES) is an intelligent agent in a cyber-physical system which cooperates with others by negotiation to fulfill a common task [14]. The advantages of CES are autonomous, scalability, changeableness and redundancy. Our case study is applied to autonomous transport robots, which is one of industrial CES. Autonomous transport robots is vehicles designed for carrying loads in storage facilities and production plants. A typical fleet consists of 4-20 robots which can carry 50-200 kg load each. Their main use is in production logistics to ensure a timely delivery and disposal of material to different sites in a production process. Given the description of a target point, the robot autonomously determines its current position, the optimal route to the destination and the maximal speed along this route. A typical usage scenario is given below:

1) An assemble station creates a job which requires a robot to deliver the packet to the target location
2) The assemble station notifies to the master control station
3) The master control station assigns the job to an idling robot
4) The robot goes to the station and loads the packet
5) The robot travels to the specific target location autonomously
6) The robot unloads the packet once the robot reaches the target location
7) The robot notifies to the master control station that the job is done

If there is no robot available, the job is postponed until some robot finishes its current job.

A practical issue shared in this realm is difficulty in estimating the optimal number of robots to achieve the desired throughput. If a road in production plants is narrow, a robot has to wait while another robot is passing through the road in order to avoid collisions. If a production plant consists of multiple floors, a robot has to check the availability of lifts and wait until one of them becomes free. Generally, as the number of robots increases, more jobs can be handled at once. However, the probability of collisions among robots also becomes higher. The throughput does not rise linearly because of overheads of the synchronization, and even overheads make the throughput worse.

B. Performance Modeling

The purposes of constructing performance model are to evaluate how many robots are necessary to satisfy its performance goals, and which of design alternatives is the most suitable. For these purposes, a model of software system should be easily compared with design alternatives under the several usage scenarios and environments where the software system is deployed. That should be taken into account in the modeling process. Our model of autonomous transport robots is composed of three kinds of submodels (shown in Fig. 3), that is, system model, usage model and resource model. Submodels keep loose coupling among each other and can be modified independently.

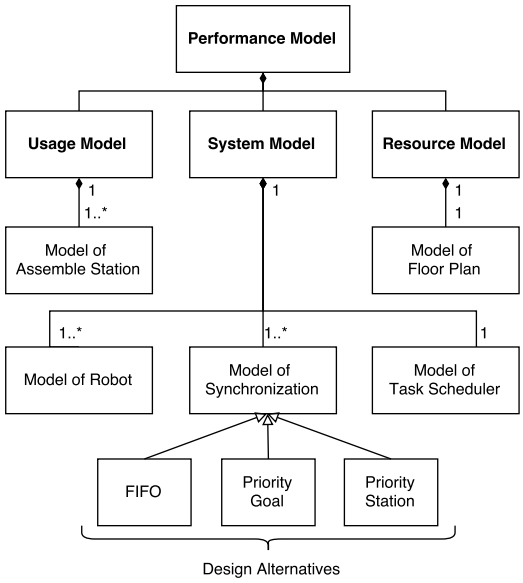

Fig. 3. The performance model structure

System model is the model of software system. Our system model comprises the behavior model of robot, the model of job scheduler and the model of synchronization between the robots. The behavior model of robot is defined based on the scenario explained in the previous section. During autonomous operation, the robot has to travel to specific target locations. Assume that each robot has the map which determines the accessible floor space and allows the robot to find the optimal route from the its current position to the target location. A factory layout floor plan is modeled on adjacency matrix. The behavior model of robot determines the optimal route by Dijkstra's algorithm so that the model can be independent from the topology of factory layout floor plan. The model of job scheduler determines the assignment of tasks to robots. Whenever a request for a transport job arrives, the job scheduler checks which robot is available. If there is more than one robot available, it is determined which of these has the best conditions. The job scheduler follows a simple first-in, first-out (FIFO) way. The model of synchronization purposes to resolve collisions between robots. When the road is narrow, only one robot can go through at a time. To avoid collisions, a robot has to wait while another robot is passing through the road. This type of synchronization mechanism has been formalized as a semaphore model. Cicirelli et al. [15] proposed a library of UPPAAL timed automata with respect to concurrent and synchronization mechanisms. We model the synchronization to realize the exclusive control on a counting semaphore automata based on the library. If there are several waiting robots, the one with the highest priority is scheduled first. The case study evaluates three variants of scheduling policy as follows:

1) FIFO
2) Prioritize robots with the packet
3) Prioritize robots without the packet

These variants of scheduling policy are considered as design alternatives.

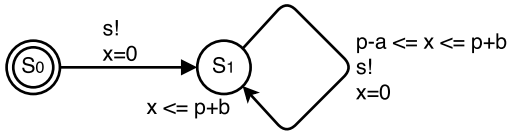

Fig. 4. Timed automata of assemble station

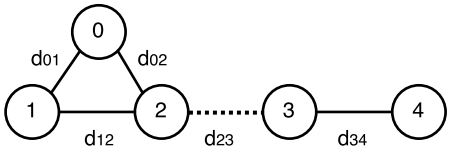

Fig. 5. Topology of factory floor plan

Usage model is the model of usage scenario and workload. Usage scenario refers a sequence to stimulate the software system and its intervals. Workload represents the number of users concurrently present in the system. In another word, usage model defines interactions between users and the software system. Our usage model is the model of assemble stations which create jobs periodically. We formalize two types of usage model: optimal usage model and jitter usage model. The optimum usage model produces a job without any jitter. After generating the first job at an arbitrary time instant, the model produces jobs periodically with a period p. In jitter usage model, on the other hand, each job (apart form the initial one) is produced within an interval $[p-a, p+b]$. The jitter usage model is formalized as non-anchord jitter automata (shown in Fig. 4) on the basis of [16]. Notice that for $a = b = 0$ we obtain a timed automata that is equivalent to the optimal usage model.

Resource model is the environment model where the software system is deployed, for example, network, servers and physical environment. Our resource model is the factory layout floor plan where the autonomous transport robots is deployed. We modeled a typical topology of floor plan, shown in Fig. 5. Each node is connected by a weighted edge, which represents distance between a node to node. Node 0 represents robot pool where all robots are allocated at the beginning. The assemble stations are placed at node 1. Target location is assigned at node 4. The road between node 2 and 3 is the critical section where only one robot can go through at a time. Robots need to wait at node 2 or 3 until other robots pass through the road.

C. Performance Metrics

As stated in Section 2, we consider three types of performance metrics, i.e., response time, throughput, and device utilization. We illustrate how to measure and verify these metrics in UPAALL SMC.

Response time refers time difference between the time when a packet is created by an assemble station and the time when a robot finish to deriver the packet to the target location. The requirement is declared like "The response time within the first 1000 time unit shall be less than X time unit in 80 percent of all requests". The response time is summation of the queueing time until the robot loads the packet, and the delivery time from the assemble station to the target location. However UPPAAL SMC doesn't allow clock values to store into local variables. We can not measure response time by this way. As an alternative solution, we create an observer automaton which capture the lifetime of a packet between the creation and the end. Ideally, the observer automaton should be created every time when the packet is created by assemble stations. However the dynamic creation of automata is not supported by UPPAAL SMC. We prepare one observer automaton and assign a lifetime of packet dynamically at a time. This solution enables to take some samples of response time through the runtime. The query over the observer automaton (ObsRes) is expressed as hypothesis testing pattern: "Pr[<=1000] ([] ObsRes.Active imply ObsRes.x <= X) >= 0.8".

Throughput refers the number of packets delivered in a certain unit. The requirement is declared like "The throughput within the first 1000 time unit shall be more than X in 90% cases". We also create an observer automaton which counts the total number of delivered packets by robots. The query over the observer automaton (ObsThr) is expressed: "Pr[<=1000] (<> ObsThr.num_packet >= X) >= 0.9".

Device utilization refers the ratio of waiting time at the critical section to total delivery time, and the ratio of idling time to total activation time. For the same reason as response time, these ratios can not be measured. Instead of using ratios, we redefine the requirements, like "The each waiting time within the first 1000 time unit shall be less than X time unit in 80 percent of all cases" and "The each idling time within the first 1000 time unit shall be less than X time unit in 80 percent of all cases". The queries are expressed: "Pr[<=1000] ([] forall(i:T) Robot(i).WAIT imply Robot(i).x <= X) >= 0.8" and "Pr[<=1000] ([] forall(i:T) Robot(i).IDLE imply Robot(i).x <= X) >= 0.8".

V. Performance Prediction and Trade-Off Analysis

We run verifications on UPPAAL SMC. The parameter settings of our usage model is shown in Table I. We allocate 6 assemble stations which create jobs every 100 time units with maximum 30 time units of jitter. As the number of robots increases, more jobs can be handled at once. However the probability of collision among robots also becomes higher. Thus the throughput does not rise linearly because of overheads of the synchronization. Even the overhead makes the throughput worse. It is important to find the optimal number of robots to achieve the performance targets. We vary the number of robots from 4 to 20 and verify whether the model satisfy each performance target. Performance requirements should be specified for the peak hour because it is that hour that matters the most in almost all applications [5]. We analyze the performance of the system at the first 1000 time unit.

TABLE I

Parameter of Assemble Stations

Parameter	Value
Number of Assemble Stations	6
Job Creation Intervals (p)	100
Jitter (a, b)	30

TABLE II

Verification Results of FIFO Scheduling Model

	Number of Robots					
Performance Property	9	10	11	12	13	15
ResTime <= 450 in 80% cases	U	U	U	U	S	S
Throughput >= 30 in 90% cases	U	S	S	U	U	S
WaitTime <= 50 in 80% cases	S	S	U	U	U	U
IdleTime <= 450 in 80% cases	S	S	S	U	U	U

Table II presents significant cases of the verification result by FIFO scheduling model. In the table, letter "S" stands for satisfied and "U" stands for unsatisfied. The result shows that there is no configuration which satisfies all performance targets. 10 robots case balances throughput and device utilization. On the other hand, 15 robots case balances response time and throughput. The results provide well-informed trade-off decision for stakeholders. The decision will be made through prioritizing performance metrics and considering other quality attributes and business considerations.

Secondly, we compare design alternatives which have different scheduling policy of synchronization. Table III shows the verification result of the model whose scheduling policy prioritize the robots with the packet. This result shows that performance is worse than the FIFO scheduling policy. The result of prioritizing the robots without the packet is almost same as Table III. These results prove that the FIFO scheduling policy is the most suitable design from the performance point of view.

VI. Related Work

In the context of software architecture, performance evaluation is known as the way of model checking, simulation and scenario-based approach.

A comprehensive overview on the model checking approach to continuous time Markov chains and Markov reward models is explained in [17]. The paper also presents typical

TABLE III

Verification Results of Prioritizing Robots with the Packet Scheduling Model

	Number of Robots					
Performance Property	9	10	11	12	13	15
ResTime <= 450 in 80% cases	U	U	U	U	S	S
Throughput >= 30 in 90% cases	U	U	U	U	U	S
WaitTime <= 50 in 80% cases	U	U	U	U	U	U
IdleTime <= 450 in 80% cases	S	S	S	U	U	U

types of performance and dependability measures in CSL. As an example of industrial scale problem using probabilistic model checking for performance evaluation, Bozzano et al. [18] defines extended AADL (Architecture Analysis and Design Language) model which aims to handle aerospace systems.

A simulation approach to evaluate component based software architectures is presented in Palladio [3]. A software architecture model formalized by Palladio Component Model (PCM) is transformed into a queueing networks for performance analysis and a discrete time Markov chains (DTMCs) for reliability analysis.

A scenario-based evaluation, known as Architecture Tradeoff Analysis Method (ATAM), is presented in [19]. ATAM is used to evaluate software architecture and identify the conflicts and trade-offs among software quality attributes by defining critical scenarios and assessing them in group meetings.

VII. Conclusions and Future Work

In this paper, we investigated the use of model checking for the analysis of non-functional requirements in industrial scale problems. We modeled an actual case study of autonomous transport robots in production plants, to experiment with stochastic model checking for the prediction of system performance. Via the case study, we addressed how to construct the performance model and how to formalize performance metrics. We ran performance verifications on UPPAAL SMC. The verification results provided well-informed trade-off decision with respect to how many robots are necessary to satisfy the performance goals, and which design alternatives is the most suitable. In the future, we plan to extended it to handle more complex systems. This paper dealt with the system comprised of homogeneous robots. Verification of a system with heterogeneous robots whose capabilities are diverse is more challenging. Our long is performance modelling and verification of decentralized AI systems (multi-agent systems), where robots interact with each other and autonomously decide upon which jobs to accept.

References

[1] Martin Glinz. A risk-based, value-oriented approach to quality requirements. *IEEE Software*, 25(2):34–41, 2008.

[2] Richard Berntsson-Svensson, Tony Gorschek, Björn Regnell, Richard Torkar, Ali Shahrokni, and Robert Feldt. Quality requirements in industrial practice - an extended interview study at eleven companies. *IEEE Trans. Software Eng.*, 38(4):923–935, 2012.

[3] R.H. Reussner, S. Becker, J. Happe, R. Heinrich, A. Koziolek, H. Koziolek, M. Kramer, and K. Krogmann. *Modeling and Simulating Software Architectures: The Palladio Approach*. MIT Press, 2016.

[4] M. A. Babar and I. Gorton. Software architecture review: The state of practice. *Computer*, 42(7):26–32, July 2009.

[5] Andr B. Bondi. *Foundations of Software and System Performance Engineering: Process, Performance Modeling, Requirements, Testing, Scalability, and Practice*. Addison-Wesley Professional, 1st edition, 2014.

[6] Nathalie Bertrand, Patricia Bouyer, Thomas Brihaye, Quentin Menet, Christel Baier, Marcus Größer, and Marcin Jurdziński. Stochastic timed automata. *Logical Methods in Computer Science*, 10(4:6), December 2014.

[7] Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008.

[8] C. Baier, B. Haverkort, H. Hermanns, and J. P. Katoen. Model-checking algorithms for continuous-time markov chains. *IEEE Transactions on Software Engineering*, 29(6):524–541, June 2003.

[9] Hans Hansson and Bengt Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6(5):512–535, 1994.

[10] Peter E. Bulychev, Alexandre David, Kim G. Larsen, Axel Legay, Guangyuan Li, and Danny Bøgsted Poulsen. Rewrite-based statistical model checking of WMTL. In *Runtime Verification, Third International Conference, RV 2012, Istanbul, Turkey, September 25-28, 2012, Revised Selected Papers*, pages 260–275, 2012.

[11] Christel Baier, Boudewijn R. Haverkort, Holger Hermanns, and Joost-Pieter Katoen. Performance evaluation and model checking join forces. *Commun. ACM*, 53(9):76–85, September 2010.

[12] Håkan L. S. Younes, Marta Kwiatkowska, Gethin Norman, and David Parker. *Numerical vs. Statistical Probabilistic Model Checking: An Empirical Study*, pages 46–60. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[13] Alexandre David, Kim G. Larsen, Axel Legay, Marius Mikucionis, and Danny Bøgsted Poulsen. Uppaal SMC tutorial. *STTT*, 17(4):397–415, 2015.

[14] B. H. Schlingloff, H. Stubert, and W. Jamroga. Collaborative embedded systems - a case study. In *2016 3rd International Workshop on Emerging Ideas and Trends in Engineering of Cyber-Physical Systems (EITEC)*, pages 17–22, April 2016.

[15] F. Cicirelli, A. Furfaro, L. Nigro, and F. Pupo. Modelling java concurrency: An approach and a uppaal library. In *2013 Federated Conference on Computer Science and Information Systems*, pages 1373–1380, Sept 2013.

[16] H. Bowman, G. Faconti, J-P. Katoen, D. Latella, and M. Massink. Automatic verification of a lip synchronisation algorithm using uppaal. In *Proc. of the 3rd International Workshop on Formal Methods for Industrial Critical Systems*, pages 97–124, 1998.

[17] Christel Baier, Boudewijn Haverkort, Holger Hermanns, and Joost-Pieter Katoen. *Automated Performance and Dependability Evaluation Using Model Checking*, pages 261–289. Springer Berlin Heidelberg, Berlin, Heidelberg, 2002.

[18] Marco Bozzano, Alessandro Cimatti, Joost-Pieter Katoen, Viet Yen Nguyen, Thomas Noll, and Marco Roveri. Safety, dependability and performance analysis of extended aadl models. *Comput. J.*, 54(5):754–775, May 2011.

[19] Paul Clements, Rick Kazman, and Mark Klein. *Evaluating Software Architectures: Methods and Case Studies*. Addison-Wesley Professional, 2001.