PetriDotNet 1.5: Configurable Stochastic Analysis Framework

[Tool Paper]

Attila Klenik
Budapest University of Technology and Economics
2 Magyar tudósok körútja
Budapest, Hungary
klenik@mit.bme.hu

Kristóf Marussy
Budapest University of Technology and Economics
2 Magyar tudósok körútja
Budapest, Hungary
kristof.marussy@inf.mit.bme.hu

András Vörös
MTA-BME Lendület Cyber-Physical Systems Reseach Group
2 Magyar tudósok körútja
Budapest, Hungary
vori@mit.bme.hu

Miklós Telek
MTA-BME Information Systems Research Group
2 Magyar tudósok körútja
Budapest, Hungary
telek@hit.bme.hu

István Majzik
Budapest University of Technology and Economics
2 Magyar tudósok körútja
Budapest, Hungary
majzik@mit.bme.hu

ABSTRACT

The emerging trend of critical systems, such as cloud, cyber-physical and safety-critical systems necessitates rigorous techniques to ensure their functionality. Extrafunctional properties such as reliability and performability are key factors and their analysis is an important part of the design process. Stochastic analysis can provide information regarding the quantitative aspects of such systems. However, many challenges have to be addressed in the application of stochastic analysis techniques. In this paper we introduce a modelling and configurable stochastic analysis framework providing a combination of various algorithms to analyse even complex stochastic systems.

Keywords

Stochastic Petri nets; Stochastic analysis; Markovian models; Numerical algorithms

1. INTRODUCTION

Critical systems are omnipresent in our everyday life. Distributed and cloud systems, cyber-physical and safety-critical systems necessitate various analysis techniques to ensure functionality and correctness. Design time analysis of the quantitative aspects of critical systems is a task important for the verification of reliability and performance requirements at an early phase of the development. However, the analysis of stochastic properties is challenging as recent real-life systems yield huge state spaces and complex stochastic behaviour. Thus, no single approach or algorithm can handle the various challenges of industrial systems.

In this paper we introduce a configurable stochastic analysis framework supporting the various tasks of the stochastic analysis process. Efficient state space traversal and storage techniques are combined with various matrix representations and numerical algorithms to support the computation of engineering measures. Beside numerical algorithms, symbolic computation of the measures is also supported, and an additional high precision numeric solver approach is also integrated into the framework. The configuration of the applied analysis workflow is based on a language which helps the definition of arbitrary analysis workflows from the given analysis building blocks. This helps the user to experiment with various settings and find the efficient configuration for their problem.

The presented tool is an extension of the PetriDotNet 1.5, which provides structural analysis, saturation-based CTL and LTL model checking and CEGAR-based reachability analysis capabilities in addition to editing and stochastic analysis support [15]. Stochastic Petri net analysis is currently only available for uncolored nets, however, other plugins can also be ran on colored well-formed nets.

PetriDotNet and our framework leverage the Microsoft .NET platform and the C♯ programming language for both the user interface and the back-end algorithms. While the use of a managed runtime for the front-end is reasonable, native code, such as C++, is more often used for model checking and numerical computation back-ends. However, in our experience, C♯ implementations can be competitive with native ones thanks to the low-level features of the language, while still being relatively safer and easier to develop. Therefore, some runtime performance is sacrificed for development time, to support tool extensions and algorithmic improvements.

ValueTools '16 October 26–28, 2016, Taormina, Italy

DOI: 10.475/123_4

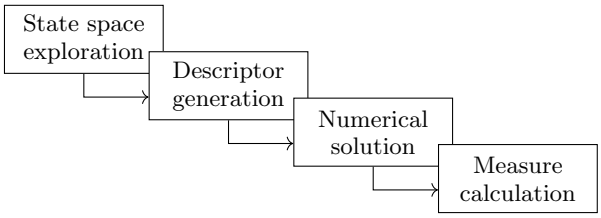

Figure 1: General analysis workflow.

2. CONFIGURABLE STOCHASTIC ANALYSIS FRAMEWORK

2.1 Supported Analyses

Stochastic analysis of complex systems requires the formulation of the engineering model to by studied as a stochastic model and its engineering measures of interest as stochastic reward measures or properties. In our framework, Markovian stochastic models are defined by the Stochastic Petri Net (SPN) formalism [5] with exponentially distributed transition firing rates that may optionally depend on model parameters, while supported measures are state and impulse reward expressions.

Our tool implements a general workflow shown in Fig. 1 that consists of

1. exploration of the possible behaviors of the system to construct its state space,
2. construction of the generator matrix of a Continous-Time Markov Chain (CTMC) based on the state space and the exponential firing times of the model,
3. numerical solution of the linear equation systems and differential equations arising from the CTMC to calculate its steady-state state distribution $\boldsymbol{\pi}$, transient and accumulated distribution vectors $\boldsymbol{\pi}(t)$ and $\mathbf{L}(t)$, as well the partial derivatives of these results with respect to the model parameters, and
4. calculation of the specified reward measures based on the numerical solutions and reward expressions specified by the user.

Reward measures can be calculated as a sum of elementary rate and impulse rewards specified by Computational Tree Logic (CTL) expressions and algebraic expressions of model parameters and the current marking. For simple problems, constant *place* rewards per token of a selected place per unit time are available as syntactic sugar, as well as constant *transition* rewards per firing of a selected transition.

The analysis workflow can calculate with steady-state and transient analysis techniques the mean reward rates $\mathbb{E}R$ in steady-state and $\mathbb{E}R(t)$ at time t. In addition, the mean accumulated transient reward $\mathbb{E}Y(t)$ is also computable. The partial derivatives of mean steady-state reward rates with respect to model parameters can be determined by sensitivity analysis. As convenience, *complex* rewards, which are algebraic expression of mean reward measures can be saved, e.g. to quickly calculate the ratio of two mean values.

Mean-time-to-state-partition, also referred to as mean-time-to-first-failure (MTFF) analysis can be used to study the reliability of systems. State partitions (fault modes) can be specified with Computational Tree Logic. The result of the analysis is the mean time to reach any of the state partitions, as well as the probability of reaching each partition first. Calculation of sensitivities to model parameter changes is also incorporated.

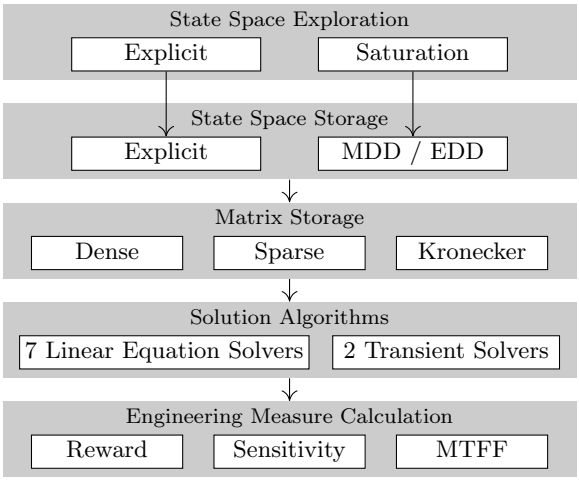

Figure 2: Available analysis components and layers.

The analyses described above can be run either standalone or as a parameter study. In parameter studies, the analysis can be ran on a grid of model parameter values.

2.2 Architecture

The stochastic analysis framework has a multi-layered architecture, where each layer is responsible for different aspects of the high-level stochastic analysis workflow described in Subsection 2.1. These responsibilities are the exploration and storage of the state space of the model, generation of the generator matrix of the associated CTMC, as well as calculation of distributions and reward measures of interest, as shown in Fig. 2.

2.2.1 Analysis Configurations

The layered architecture allows the user to combine various implementations of the stochastic analysis tasks. One can choose the stochastic analysis workflow according to the characteristics of the input model to be analyzed and the available computation resources. 147 analysis combinations are possible in total, which shows the high flexibility and configurability of the framework.

The state space exploration layer consists of our implementation of explicit state space exploration and the *saturation* algorithm [3]. Saturation is a symbolic state space exploration method that constructs a multi-valued decision diagram (MDD) representation for the state space.

The generator matrix Q of the continuous-time Markov chain associated with the stochastic Petri net may be stored as a dense or sparse matrix. In addition, the block Kronecker form [1] is also available, which may achieve significant memory savings with little run time overhead if the model is appropriately partitioned.

The construction of dense and sparse matrices from the explicit or symbolic state space and the stochastic behaviors of the model is straightforward. On the other side, block Kronecker matrices first require state space decomposition.

This state space decomposition, as well as mapping between the original and decomposed state indices of the model, is supported for both explicit [1] and symbolic [6] state space representations. Index mappings are represented as edge-valued decision diagrams (EDDs) [11].

After the successful construction of the matrix representations, the linear equation systems and differential equations arising from the stochastic analysis task can be solved by various numerical algorithms. Some algorithms, for example, block iterative linear equation solvers and implicit integrators for differential equations delegate sub-tasks to other numerical algorithms, which is also user-selectable.

The output of the numerical algorithms are vectors corresponding to the state distributions of the systems and in case of sensitivity analysis, its partial derivatives with respect to model parameters. To extract the relevant engineering measures from these objects, post-processing algorithms are available to compute instantaneous and accumulated reward measures, their sensitivity values, as well as mean-time-to-first-failure (MTFF) values and failure mode probabilities.

To execute the analysis, the user must select a state space exploration and generator matrix storage method to construct the corresponding CTMC of the system. In addition, the numerical algorithm used to compute the solution vectors must be specified. The numerical algorithms are executed and their results are post-processed automatically according to the reward measures which are to be calculated by the analysis.

2.2.2 Linear Algebra Library

Efficient vector and matrix storage, as well as efficient linear algebra operations are a cross-cutting concern in the layered stochastic analysis framework. Numerical solution algorithms perform low-level operations, such as vector addition, scalar products and vector-matrix products on the state probability vectors and generator matrices.

Louse coupling between the solver implementations and the matrix storage techniques was an important goal during the design of the linear algebra functionalities. In addition, the user can select between different ways of performing elementary operations, for example, the choice between parallel and sequential evaluation of vector-matrix products. This way we extended the configurability further.

To our best knowledge, no existing .NET linear algebra library provides the ability to work with complex expression of matrices that arise in the decomposition of CTMC generators as if they were ordinary matrices. Therefore, we implemented our own library in the framework on the pillars of *expression trees* and *multiple dispatch*.

Large matrices are stored as expression trees, where the leaves of the tree may be dense or sparse matrices or vectors corresponding to diagonal matrices. The inner nodes are linear algebra operators over which vector-matrix products commute: splitting a matrix into blocks, linear combinations and Kronecker products. A similar type hierarchy is introduced for vectors to support the splitting of vectors into blocks and as an optimization for special vectors, for example those that contain a single nonzero element.

Recall that block Kronecker decompostion expresses the generator matrix Q in the form $Q = Q_O - \operatorname{diag} \mathbf{q}_D$. The off-diagonal part is split into blocks $Q_O = \big(Q_O[\tilde{x}, \tilde{y}]\big)_{\tilde{x},\tilde{y}=0}^{\tilde{n}-1,\tilde{n}-1}$. In turn, each block is a linear combination of Kronecker products $Q_O[\tilde{x}, \tilde{y}] = \sum_t \big(\lambda_t \cdot \bigotimes_j Q_t^{(j)}[\tilde{x}, \tilde{y}]\big)$, where the factors are either sparse or identity matrices. The expression tree approach allows the calculation with the decomposed matrix form and without explicitly noting that it is a complex expression, as it only uses operators which can be inner tree nodes (Fig. 3). Any decomposition of matrices that utilizes these operations can be realized with our approach.

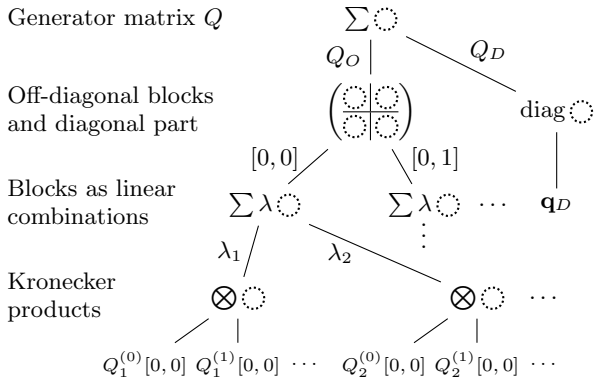

Figure 3: Block Kronecker decomposition tree.

Each basic operation, e.g. vector addition, scalar product, vector-matrix product is realized as a multimethod that dispatches to the concrete implementation based on the dynamic types of its arguments. The SHUFFLE algorithm [2] is used for multiplication with Kronecker products of matrices.

Implementations of the elementary operations heavily rely on the `unsafe` facilities provided by the C♯ language for direct memory access and manual memory allocation to avoid performance overhead associated with some .NET features. The multiple dispatch logic also simplified development of specialized operation implementations for otherwise problematic combinations of input objects that may be discovered by profiling, as dispatch rules can be added for the specialized operations.

Further configurability is achieved by replacing the dispatch logic at runtime to switch between parallel or sequential implementations. If the numerical solution algorithm can take advantage of high-level task-based parallelism, sequential linear algebra operations are preferable. However, for sequential solvers, low-level elementary operations can be evaluated on multiple cores simultaneously.

2.3 Numerical Algorithms

Several numerical solvers are available in our framework. Implementations were done from scratch to support the expression tree based generator matrices.

Steady state and mean-time-to-first-failure measures are calculated by solving usually sparse systems of linear equations. Suitable solution algorithms in our framework are

- direct solution of the linear equation system by *LU decomposition*,
- basic *power, Jacobi* and *Gauss–Seidel* stationary iterative solvers with optional overrelaxation,
- *group* versions of *Jacobi* and *Gauss–Seidel* methods, in which subproblems arising from blocks along the diagonal of the generator matrix are solved by inner linear equation solver, and

- the *BiCGSTAB* algorithm, which is a popular Krylov subspace method that offers a good compromise between convergence criteria, run time and memory requirements.

For a description of these solvers, we refer the reader to [13, Chapters 4 and 7].

Transient solutions of CTMCs for immediate and accumulated distribution vectors are provided by the well-known *uniformization* (or *randomization*) algorith. In addition, the *TR-BDF2* implicit integrator is also part of the framework to handle *stiff* Markov chains that may take many iterations to solve with uniformization [10]. Being an implicit method, TR-BDF2 requires the selection of a linear equation solver for the arising subproblems.

The built-in algorithms and configuration options allow the user to customize the analysis according to the model to be studied and the available computational resources. However, the outcome of the customization i.e. the performance of the constructed configuration relies on the developers experience and knowledge about the models/algorithms. We performed some initial experiments on stochastic versions of Petri nets from the Model Checking Contest[1] in [7]. Our measurements indicated that for steady-state solutions, BiCGSTAB is usually sufficient, but surprisingly, Gauss–Seidel iteration may be faster on some models. Moreover, group Gauss–Seidel iteration may reduce memory consumption when solving systems of equations with block matrices. Note that the memory bottleneck with block Kronecker decomposed generator matrices is mainly formed by the temporary vectors to be stored; BiCGSTAB uses 7 temporary vectors, while group Gauss–Seidel needs only one.

We also attempted to adapt the Krylov subspace solver IDR(s)STAB(ℓ) into our framework. By choosing the parameters s and ℓ appropriately, IDR(s)STAB(ℓ) becomes a common generalization of several other methods, including BiCGSTAB(ℓ) and IDR(s) [14]. Thus, the run time and memory tradeoff can be tuned according to the user's preference. However, despite our modifications to the algorithm in [7], convergence behavior with CTMC generator matrices remained unsatisfactory, as their rank deficiency seems to pose an obstacle for the algorithm.

2.3.1 Symbolic Evaluation

As an extension of the expression tree approach for matrices, we added support for storing *elements* of the matrices and vectors involved in the analysis as algebraic expression trees. Instead of fixed values, mean reward measures are obtained as symbolic functions of model parameters.

In contrast with methods specific to parametric Markov chains, such as [4], in principle any analysis involving linear equation solvers can be adapted. Our tool supports symbolic solution for mean steady state reward values and mean times to reach state partitions.

Unfortunately, linear equation solvers can only depend on the nonzero structure of the matrices, as the numerical values of model parameters and hence the matrix elements themselves must be treated as unknowns. Therefore, only direct linear equation methods are available and no pivoting can be used. In our tool, symbolic solution is carried out by LU decomposition.

To reduce memory requirements, expression trees need to be simplified by algebraic manipulations and constant folding during analysis. The user can select at which points should this simplification happen.

2.3.2 High-precision Evaluation

Because pivoting and other techniques to improve numerical stability are unavailable in symbolic evaluation, catastrophic cancellation and underflow may occur during constant folding. To alleviate these problems, constants in expression trees can be represented by a .NET arbitrary-precision floating-point library[2] in addition to double-precision floating-point values.

For use in specialized scenarios, high-precision arithmetic is also supported in purely numerical evaluation. In this mode, matrix and vector elements are arbitrary-precision numbers and any linear equation solver can be ran. Perharps surprisingly, we found no improvement of the convergence behavior of numerical algorithms when ported to the arbitrary-precision data structures compared to ordinary double-precision arithmetic. Therefore, this mode of operation is only of limited use in problems where many significant digits of mean reward values are desired.

2.4 Extensibility

PetriDotNet 1.5 has a plugin system that provides access to both Petri net data structures and the graphical user interface. The extensibility features have made PetriDotNet an analysis platform for Petri nets thanks to the continuous interest of developers, especially M.Sc. and Ph.D. students at our university.

The stochastic analysis plugin allows users to execute analyses provided by our framework using the graphical user interface of PetriDotNet. However, stand alone operation as a command line application and inclusion into other programs as a library is also possible.

Therefore, to facilitate easy incorporation of new algorithms into the framework while allowing independent use, our framework has an extensibility mechanism orthogonal to the plugin system of PetriDotNet 1.5.

Algorithms are implemented as C♯ classes with inputs and outputs marked with attributes that carry meta-information, such as marking an input as optional. In addition, cross-cutting concerns, including cancellation, logging and linear algebra subroutines can be injected as dependencies. The algorithms are organized into layers as described in Subsection 2.2, while uniform interfaces implemented by the algorithms allow easy replacement of components from each layer within the analysis workflow.

3. INTERFACES

3.1 PetriDotNet 1.5 Plugin

A plug-in was created for the PetriDotNet 1.5 Petri net editor that exposes stochastic analysis functionalities in the user interface. Mmodel parameters, exponential transition rates, reward expression and fault modes are saved as part of the PNML Petri net files (Fig. 4). analysis configurations, i.e. the selection and parameters for the analysis algorithms, can be saved and executed in the Net Analysis toolbox (Fig. 5).

[1] http://mcc.lip6.fr/models.php

[2] https://peteroupc.github.io/Numbers/

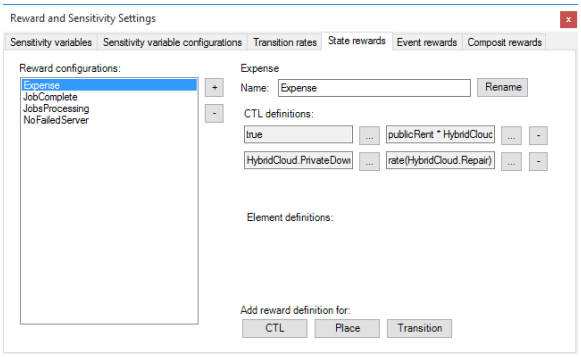

Figure 4: The reward definition interface.

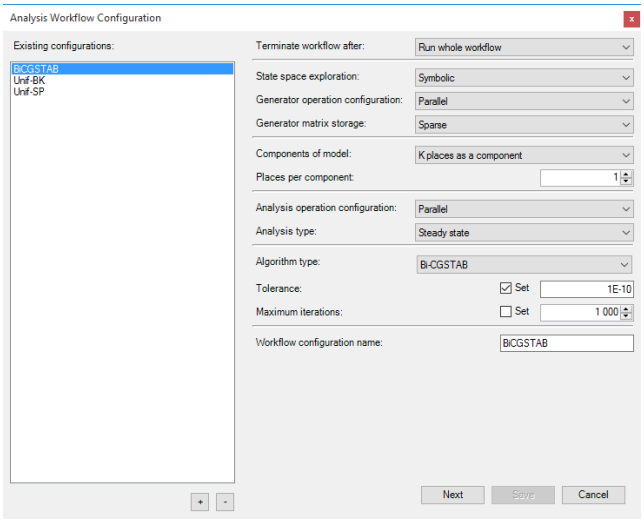

Figure 5: Analysis configuration and execution.

The installation and usage of PetriDotNet is extremely simple. The tool with the stochastic analysis framework, as well as its user manual can be downloaded from our website[3]. After download, the tool can be started by running PetriDotNet.exe.

A set of example and benchmark models is distributed with the tool, located in the models folder. Further information and models are available at the website of the stochastic analysis extensions[4].

3.2 Lightweight Analysis Workflows

Connecting algorithms according to the stochastic analysis workflow may become difficult to manage due to the interdependencies of the solution algorithms. For example, group iterative linear equation solvers can be only used with block structured generator matrices. Additionally, some algorithms are only suitable for steady-state solutions, but do not converge for sensitivity and mean-time-to-first-failure problems. Especially for ad-hoc analyses, such as computing mean-time-between-failures, creating a user interface for configurable analysis tasks can be cumbersome.

As a proof of concept, we developed a *lightweight* analysis workflow engine for our framework. Dynamic creation and execution of workflows is demonstrated through a client Eclipse plugin that accesses a web service realized as a Windows Communication Foundation (WCF) endpoint interpreting the workflow definitions. However, note that the proof of concept tools are not bundled with the public release of PetriDotNet 1.5 due to their currently volatile interfaces.

[3]http://petridotnet.inf.mit.bme.hu/en

[4]http://inf.mit.bme.hu/en/petridotnet/stochasticanalysis

```
<Workflow xmlns="http://petridotnet.inf.mit.bme.hu/..."
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:xlinq="clr-namespace:System.Xml.Linq;...">
  <Workflow.Inputs>
    <Input Name="Pnml" Type="xlinq:XDocument" />
  </Workflow.Inputs>
  <Workflow.Outputs>
    <Output Name="Size"
        Value="{Wire Result.Size, SourceName=Edd}" />
  </Workflow.Outputs>
  <!-- Three algorithms are executed sequentially. -->
  <Pdn15StochasticPnmlLoaderAlgorithm />
  <Pdn15KmSaturationAlgorithm />
  <MddStateSpaceEnumeratorAlgorithm x:Name="Edd" />
</Workflow>
```

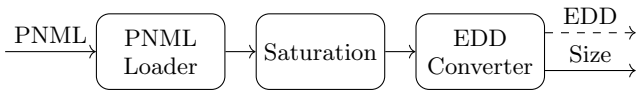

Figure 6: Example XAML analysis workflow.

Analysis algorithms to be executed can be specified in XAML, an XML-based object graph description language developed by Microsoft. The XAML workflow may also declare inputs, outputs and explicit data flow between algorithms. However, most data flow is inferred implicitly based on .NET static typing rules. Editor support, such as autocompletion, is provided by the XAML editing capabilities of Microsoft Visual Studio.

An example configuration, which only uses the state space exploration and storage layers, is shown in Fig. 6. Some XML namespaces were abbreviated due to lack of space. Note that only the data flow to the output is explicit, the rest can be inferred from the algorithm interfaces.

In contrast to orchestration solutions for model transformation and verification, such as the SENSORIA Development Environment [8], our lightweight XAML solution is .NET-specific, hence not suitable for large-scale tool integration. However, the creation of ad-hoc combinations of algorithms, including integration tests and custom analyses leveraging the PetriDotNet platform, can be still simplified greatly in comparison to manual C♯ glue code.

4. CASE STUDY

The application of our stochastic analysis framework in an industrial project is shown. This case-study served as a motivation to implement the new analysis methods in the framework and helped us to evaluate its usability.

Part of the project was to derive safety measures for a feature of an automotive system whose design employs dual modular redundancy and self-checking mechanisms for fault-tolerance. The subsystem was modelled using the SPN formalism and the critical set of states were described by temporal logic formulas over the places of the net. The possible events that can occur during operation, e.g. component faults, were mapped to transitions with exponentially distributed firing rates. The rates of events heavily relied on

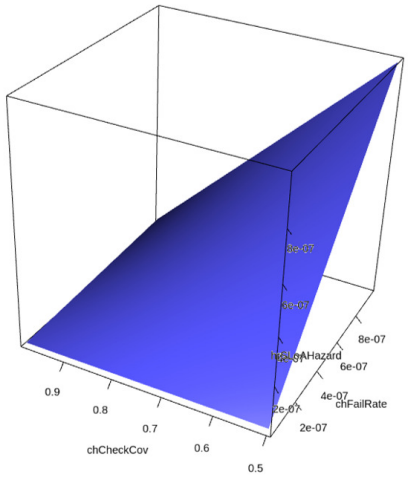

Figure 7: Visualization created with Shiny.

the model parameter support of our framework which facilitated the quick reconfiguration of the model.

Exploring and storing the state space and stochastic behavior of the acquired model was feasible due to its manageable size. However, the conventional methods for steady-state and transient analysis were not enough to derive the necessary safety metrics. The measures of interest included the mean-time-to-first-hazard (MTFH) for a specified hazard, its corresponding hazard rate, which is the reciprocal of the MTFH value, and the sensitivity of these mean time values to some model parameters.

To our best knowledge, our framework is the only non-commercial, freely available tool that, besides instantaneous and accumulated mean reward value calculation, also supports the *automatic* calculation of mean-time-to-state-partition values and their sensitivities to model parameters. Combining this analysis feature with the quick reconfigurability of the model due to model parameter support, we were able to perform an automatic parameter study deriving the measures of interest for a wide value range of selected variables. This resulted in around one million measure calculations in a predefined subspace of the parameter space. The acquired results were used to visualize and analysed the safety metrics of the subsystem in an interactive web browser interface (Fig. 7) based on R [9] and Shiny [12].

As our case-study shows, the development of a configurable framework gives huge potential for extending the set of solvable problems. Configurability provides the user with the ability to fine-tune the analysis and supports the experimentation or it can serve as a portfolio solver for huge problems. In the future we follow this way and we plan to further extend the framework with new algorithms and also we plan to support a wider range of stochastic modelling formalisms

5. ACKNOWLEDGMENTS

We would like to thank Prof. Gianfranco Ciardo for his comments and valuable advice on high-precision evaluation.

This work was partially supported by the ARTEMIS JU and the Hungarian National Research, Development and Innovation Fund in the frame of the R5-COP project.

This research was partially performed within the framework of the grant of the Hungarian Scientific Resarch Fund (grant no. OTKA K101150).

6. REFERENCES

[1] P. Buchholz. Hierarchical structuring of superposed GSPNs. *IEEE Trans. Software Eng.*, 25(2):166–181, 1999.

[2] P. Buchholz, G. Ciardo, S. Donatelli, and P. Kemper. Complexity of memory-efficient Kronecker operations with applications to the solution of Markov models. *INFORMS Journal on Computing*, 12(3):203–222, 2000.

[3] G. Ciardo, Y. Zhao, and X. Jin. Ten years of saturation: A Petri net perspective. *ToPNoC*, 5:51–95, 2012.

[4] E. M. Hahn, H. Hermanns, and L. Zhang. Probabilistic reachability for parametric markov models. *STTT*, 13(1):3–19, 2011.

[5] M. A. Marsan. Stochastic petri nets: an elementary introduction. In *Advances in Petri Nets 1989*, pages 1–29, 1988.

[6] K. Marussy, A. Klenik, V. Molnár, A. Vörös, I. Majzik, and M. Telek. Efficient decomposition algorithm for stationary analysis of complex stochastic Petri net models. In *PETRI NETS 2016*, pages 281–300, 2016.

[7] K. Marussy, A. Klenik, V. Molnár, A. Voros, M. Telek, and I. Majzik. Configurable numerical analysis for stochastic systems. In *2016 International Workshop on Symbolic and Numerical Methods for Reachability Analysis (SNR)*, pages 1–10. IEEE, 2016.

[8] P. Mayer and I. Ráth. The sensoria development environment. In *Rigorous Software Engineering for Service-Oriented Systems – Results of the SENSORIA Project on Software Engineering for Service-Oriented Computing*, pages 622–639. 2011.

[9] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2008. ISBN 3-900051-07-0.

[10] A. Reibman, R. Smith, and K. Trivedi. Markov and Markov reward model transient analysis: An overview of numerical approaches. *European Journal of Operational Research*, 40(2):257–267, 1989.

[11] P. Roux and R. Siminiceanu. Model checking with edge-valued decision diagrams. In *NFM 2010*, pages 222–226, 2010.

[12] RStudio, Inc. *Easy web applications in R.*, 2013. URL: http://www.rstudio.com/shiny/.

[13] Y. Saad. *Iterative methods for sparse linear systems*. Siam, 2003.

[14] G. L. Sleijpen and M. B. Van Gijzen. Exploiting BiCGstab(ℓ) strategies to induce dimension reduction. *SIAM journal on scientific computing*, 32(5):2687–2709, 2010.

[15] A. Vörös, D. Darvas, V. Molnár, A. Klenik, Á. Hajdu, A. Jámbor, T. Bartha, and I. Majzik. Petridotnet 1.5: Extensible petri net editor and analyser for education and research. In *PETRI NETS 2016*, pages 123–132, 2016.