DiffLQN: Differential Equation Analysis of Layered Queuing Networks

Tabea Waizmann Mirco Tribastone
IMT — School for Advanced Studies
Lucca, Italy
{tabea.waizmann,mirco.tribastone}@imtlucca.it

ABSTRACT

Layered queuing networks are a popular technique in software performance engineering. In this paper we present DiffLQN, a tool for the analysis of networks using ordinary differential equations. It estimates average performance indices such as throughput, utilization, and response time of software and hardware devices. The complexity of computing the solution is independent of the concurrency levels in the model (i.e., thread multiplicities and processing units) and the estimates are theoretically guaranteed to be asymptotically correct for large enough concurrency levels. DiffLQN is designed having in mind compatibility with other tools that support state-of-the-art methods based on mean value analysis.

CCS Concepts

•General and reference → Performance; •Software and its engineering → Software performance;

Keywords

Layered queuing networks; Ordinary differential equations; PEPA

1. INTRODUCTION

Layered Queuing Networks (LQN) are a popular model in software performance engineering because they support, as first-class citizens, frequently used high-level mechanisms such as synchronous and asynchronous communication, layered services (i.e., entities that work as clients as well as servers), and fork/join synchronization [6]. In addition to stochastic simulation, LQNs are traditionally solved analytically using approximate mean value analysis (AMVA), an efficient technique that provides estimates of steady-state performance indices such as throughput, utilization, and response time. The approximation comes from two main sources: first, a recursive algorithm depending on the number of jobs in the network is replaced by a fixed-point iteration independent on them, e.g., [4]; second, heuristics are used to capture non-product-form behavior in the AMVA framework, e.g., [15].

Although the approximation has been empirically shown to be satisfactory, no guarantees can be provided [13]. To partially mitigate this problem, more recently an alternative interpretation of LQNs in terms of ordinary differential equations (ODEs) has been proposed [17, 18, 19]. This has been motivated by (unrelated) results on ODE approximations for process algebra models, specifically for PEPA [9, 24, 20]. In this view, an ODE is associated with every basic activity in the LQN which provides an estimate of the average number of jobs executing that activity. Notably, building on a fundamental result by Kurtz on fluid limits of Markov population processes [12], the approximation has an asymptotic guarantee of exactness when the multiplicity of jobs and servers is large enough. This is in contrast to the error behavior of AMVA, which may even increase with larger multiplicities [19]. A useful side product of the ODE approach is that it also readily gives indices of performance for the transient behavior. Indeed, while AMVA only considers the steady state, this is computed by running a transient analysis for long enough until convergence to a fixed point.

This paper presents DiffLQN, a software tool that supports ODE analysis of LQNs. DiffLQN is 100% Java. A single-JAR executable is available for download at http://sysma.imtlucca.it/tools/difflqn/.

With this contribution we aim to address the following software performance engineering challenges:

- We reduce the effort in applying performance methods that use formal languages as the underlying engine. Indeed, the main design rationale behind the tool development was to hide the process algebra to the end user, who is exposed to the more common vocabulary of LQNs. This can be further bridged to higher-level specifications such as performance-annotated UML diagrams (e.g., [25, 22, 23]).
- With formal asymptotic guarantees of correctness, we aim to increase the modeller's confidence to interpret and use the performance results obtained through the ODE analytical solver.
- From a practical viewpoint we ease (and encourage) tight integration with other software modeling tools, by providing a self-contained application that requires no further libraries or execution environments than a Java virtual machine.

ICPE'16 Companion, March 12-18, 2016, Delft, Netherlands

 ISBN 978-1-4503-4147-9/16/03…$15.00
DOI: http://dx.doi.org/10.1145/2859889.2859896

2. RELATED WORK

The state-of-the-art tool for LQNs is Carleton University's LQNS [3]. It offers simulation and analytical computation of performance measures for LQNs, using AMVA-based algorithms. To facilitate easy usability of DiffLQN, we designed it having in mind compatibility with LQNS, supporting a text-based syntax that presents only minor deviations. Support for the alternate XML-based input format used by LQNS will be implemented in the future.

The Palladio-Bench is an Eclipse-based integrated modelling environment and uses the Palladio Component Model (PCM) [2] to predict performance measures. A PCM-to-LQN model transformation was introduced in order to make use of efficient solving strategies for LQNs to analyze Palladio models [11]. The Palladio-Bench now includes a module for automatic translation of PCM instances into LQNs. It would be possible to use DiffLQN to mediate the transformation from PCM to LQN, in this way providing another alternative, ODE-based, analysis method for PCMs.

With LINE [14], another tool for ODE-based analysis of queuing networks has recently been developed. LINE supports processor-sharing scheduling, whereas DiffLQN assumes a FCFS scheduling. Unlike DiffLQN it supports percentile analysis and can output ODE transient performance indices. However, it does not yet cover a number of core features of the LQN model, such as asynchronous calls, task multiplicities, fork/join synchronization nodes, and activities with second phases (early replies). All these features are available in DiffLQN. Finally, regarding the user interface, LINE depends on MATLAB, while DiffLQN is a Java application. LINE accepts LQNs in the XML format, while DiffLQN uses the textual syntax of LQNs.

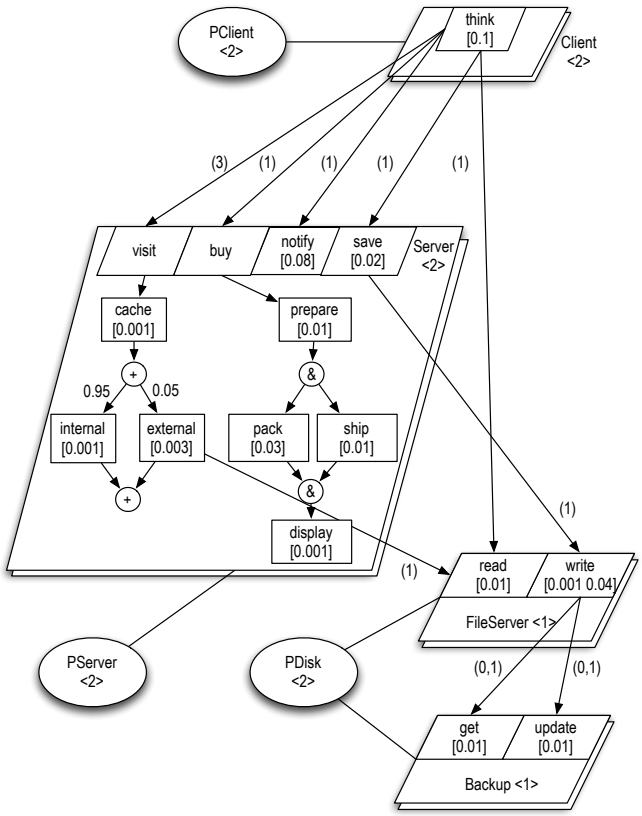

Figure 1: Graphical representation of an LQN, taken from [19].

3. OVERVIEW OF LQNS

In order to make this paper self-contained, we start with a brief overview of LQNs using an example that exhibits all the features supported by DiffLQN. We refer to [6] for the details on LQNs. Our sample LQN is graphically depicted in Fig. 1 (although we remark that this is specified concretely in a text format). The basic computational resource is a *processor* (ovals) on which *tasks* (large parallelograms), for instance software services, are deployed. A task consists of different *entries* (smaller parallelograms) that represent distinct kinds of services. An entry can be a basic *activity*, the atomic unit of operation in LQN if it is not further specified. Otherwise, it points to a diagram of basic activities (rectangles) which are performed in sequence (linked by an arrow), through probabilistic choice via *decision/merge* nodes ('+' operator), or by means of fork/join synchronization ('&' operator).

Activities can call entries synchronously (closed arrowhead) or asynchronously (open arrowhead, not shown in example). Asynchronous requests just start the requested entry while continuing the current activity, without waiting for a reply. In synchronous requests, the activity is stopped until a reply arrives. An activity can send any number of requests to the same entry; the number of requests is written next to the call arrow in brackets. When called, an activity consumes time on the processor where the task is deployed. Time demands are shown below the activity name within square brackets. When two time demands are listed, as in write, second-phases are modeled. The second time demand happens after the activity has returned control to the caller, effectively executing asynchronously.

4. DIFFLQN

4.1 Architecture

Currently, the front-end of DiffLQN is a command line that accepts a text-based representation of an LQN. The parser is automatically generated from Eclipse's Xtext framework. In [19], the algorithmic derivation of the ODEs was mediated by a translation of an LQN into a model written in the process algebra PEPA [8]. We exploit this fact, by converting the abstract syntax tree of an LQN into a PEPA model in order to leverage the tool support for this process algebra: in particular, we use PEPAto, the API of the PEPA Eclipse Plugin [21] in order to generate, analyze, and manipulate the ODEs of the LQN model.

In essence, DiffLQN tracks the correspondence from LQN elements to process-algebra models, and back for the propagation of the analysis results. In this way, the intermediate translation step into PEPA is hidden to the end user.

4.2 Capabilities

DiffLQN provides the following LQN performance indices:

- *Throughput*, at different levels of granularity: it provides the average number of activities, entries, or tasks completed per unit time at steady state.

- *Utilization* for processors and tasks, giving the average number of busy entities at steady state. Utilization estimates are also provided per single entry and activity, giving their contribution to the utilization of the processor on which they are deployed.

- *Response time*, the average response time at steady state for the execution of entries or tasks.

These can be computed by numerical integration of the ODEs or by simulation. Because of the presence of very fast rates in the LQN encoding presented in [19], we extended PEPAto with a stiff solver, adapted from the BioUML workbench [10]. Stochastic simulation is implemented using Gillespie's algorithm [7], directly leveraging the implementation available in the PEPA Eclipse plug-in.

4.3 Syntax

Model specification. To favour compatibility, DiffLQN accepts a slight variant of the text-based input format for LQNS. We refer to [3] for a complete documentation on the grammar. Naming conventions specific to DiffLQN are described on our website.

Some LQN features, e.g. loops and tasks with infinite multiplicity, are not currently supported. The input file template that is available on the website explains all supported keywords. DiffLQN accepts LQN models containing unsupported features as valid syntax, but the solver either emits a warning, or explains the problem in an error message.

Fig. 2 shows the input file for the network of Fig. 1. Briefly, the `G`-block in lines 1–7 is ignored by DiffLQN because it provides parameter settings specific to LQNS. Processors are defined in lines 11–13 within the block `P 0` and `-1`. Keyword `m` declares their multiplicity. Tasks are defined in a similar fashion (lines 17–22). Each line declares the list of entries running on the task. Keyword `r` declares *reference tasks*, i.e., tasks that do not accept requests (to model jobs/clients). Entries are specified in lines (26–42). Service demands are given in lines starting with `s`, calls are specified by the keyword `y` (synchronous) and `z` (asynchronous). Activities are declared using keyword `A` and are further specified in block (46–61). Finally, options specific to DiffLQN are in lines 65–70.

Solver settings. ODE analysis is performed by solving an initial value problem numerically until convergence to steady state is detected (or if a threshold time horizon is reached, in which case a warning is issued if convergence has not been reached). The convergence criteria are based on *absolute* and *relative tolerances*. The former considers the Euclidean norm of the derivatives of the solution at the current time point, and absolute convergence is reached when this value is below a given threshold (formally, the norm must be equal to zero in the steady state); the latter compares the norm of the difference between the solutions at successive time points. By default, the analysis terminates successfully when both the absolute and the relative convergence criteria are met.

Stochastic simulation is performed using the method of batch means [16]: roughly speaking, a single simulation run is performed and statistics are collected across different non-overlapping parts of the run (the batches) which are assumed to be long enough that the system has reached steady state.

```
1  G
2  "Example-LQN"
3  0.0001
4  500
5  1
6  0.5
7  -1
8
9  # processor definition block
10 P 0
11 p PClient f m 2
12 p PServer f m 2
13 p PDisk f m 2
14 -1
15
16 # task definition block
17 T 0
18 t Client r think -1 PClient m 2
19 t Server n visit buy notify save -1
       PServer m 2
20 t FileServer n read write -1 PDisk
21 t Backup n get update -1 PDisk
22 -1
23
24 # entry definition block
25 E 0
26 s think 0.1 -1
27 y think visit 3.0 -1
28 y think save 1.0 -1
29 y think notify 1.0 -1
30 y think read 1.0 -1
31 y think buy 1.0 -1
32 A visit cache
33 A buy prepare
34 s save 0.02 -1
35 y save write 1.0 -1
36 s notify 0.08 -1
37 s read 0.01 -1
38 s write 0.001 0.04 -1
39 y write get 0.0 1.0 -1
40 y write update 0.0 1.0 -1
41 s get 0.01 -1
42 s update 0.01 -1
43 -1
44
45 # activity definition block for task
       Server
46 A Server
47  s prepare 0.01
48  s pack 0.03
49  s ship 0.01
50  s display 0.001
51  s cache 0.001
52  s internal 0.001
53  s external 0.003
54  y external read 1.0
55 :
56  prepare -> pack & ship;
57  pack & ship -> display;
58  cache -> (0.95)internal + (0.05)
        external;
59  internal[visit];
60  external[visit];
61  display[buy]
62 -1
63
64 # DiffLQN settings block
65 #! v 1.0e5
66 #! solver sim
67 #! confidence_level 0.98
68 #! confidence_percent_error 2.0
69 #! stoptime 1000.0
70 #! export csv
```

Figure 2: Example input file for the network of Fig. 1.

	Scenario x1			*Scenario x10*			*Scenario x20*		
Metric/Kind/Name	ODE	Sim.	Error	ODE	Sim.	Error	ODE	Sim.	Error
Th / act / prepare	7.685	6.741	14.01	76.854	76.490	0.48	153.708	153.544	0.11
Th / act / external	1.153	1.010	14.18	11.528	11.391	1.21	23.056	22.945	0.49
Th / entry / visit	23.056	20.282	13.68	230.561	229.743	0.36	461.123	460.283	0.18
Th / task / Server	46.112	40.491	13.88	461.123	459.580	0.34	922.245	921.388	0.09
Th / task / Backup	15.371	13.491	13.93	153.708	152.976	0.48	307.415	307.535	0.04
Ut / proc / PClient	0.769	0.674	13.98	7.685	7.663	0.29	15.371	15.357	0.09
Ut / task / Server	1.142	1.132	0.90	11.418	11.534	1.01	22.835	23.034	0.86
Ut / task / Backup	0.154	0.135	13.93	1.540	1.533	0.48	3.080	3.081	0.04
PU / act / pack	0.231	0.203	13.81	2.306	2.309	0.14	4.611	4.611	0.00
RT / entry / think	0.260	0.297	12.26	0.260	0.261	0.29	0.260	0.260	0.09
RT / entry / save	0.021	0.031	32.29	0.021	0.021	0.65	0.021	0.021	0.03
RT / entry / notify	0.080	0.080	0.24	0.080	0.080	0.00	0.080	0.080	0.00
RT / entry / visit	0.002	0.003	34.10	0.002	0.003	18.07	0.002	0.003	15.61
RT / task / Server	0.025	0.028	11.40	0.025	0.025	1.34	0.025	0.025	0.96
RT / task / FileServer	0.034	0.034	0.02	0.034	0.034	0.02	0.034	0.034	0.11
Average percentage errors			**13.51**			**1.67**			**1.25**

Table 1: Numerical results, internal comparison of DiffLQN.

Solver settings for DiffLQN are backward compatible with LQNS since every line must start with '`#!`', which is treated by LQNS as a comment. Below we list the settings that are currently supported.

- `v` specifies the value for a fast rate `v` that approximates the behavior of certain operations, such as forks and joins, that are assumed to be instantaneous in LQNs. This is the only mandatory setting.
- `solver [ode | sim]` specifies whether to use ODE analysis or stochastic simulation.
- `stoptime` specifies the maximum time horizon for the numerical ODE integration or the length of an initial transient simulation run that is removed before batch statistics are collected.
- `solver_abs_tol` and `solver_rel_tol` are typical absolute and relative tolerances for the ODE numerical integration [1].
- `steady_abs_tol` and `steady_rel_tol` specify the tolerances for ODE steady-state detection, as discussed above.
- `[ absolute | relative ] steady state` is a flag for using only one of the two criteria of steady-state convergence.
- `batch_length_factor` specifies the length of a batch, relative to the initial transient defined with `stoptime`.
- `confidence_level`/`confidence_percent_error` specify the usual termination criteria for stochastic simulation.

Output settings. By default, DiffLQN computes all possible performance measures discussed in Sect. 4.2. Optionally the user can explicitly choose which measures to track. This can be speed up the computation, especially for large networks analyzed using simulation [20]. This is done in a block with lines starting (in order) with keywords `throughput`, `utilisation`, and `response time`, followed by a list of desired elements for the respective performance index.

Exporting options. By default, analysis results are outputted to the screen in a human-readable format. However, the LQN model as well as the results can be exported in different formats. Each export command is specified in a new line with the `export` keyword, followed by the type of export requested (and an optional file path). Available export types are:

- `pepa`: Export of the PEPA encoding of the input LQN, in a format that is compatible with the PEPA Eclipse plug-in.
- `matlab`: A function file in Matlab-compatible form which can be used in conjunction with Matlab's ODE solvers (e.g., the stiff solver `ode15s`)
- `csv`: Results are saved to a comma-separated values file.

5. CASE STUDY

As a case study we evaluate DiffLQN on the running example. For this, we consider a comparison between the ODE results and the simulation results. The latter are taken to be the "true" values of the performance indices, following the successful validation against the simulation results of LQNS performed in [19]. In particular, the settings in Fig. 2 indicate that the simulations were set to stop when the 98% confidence levels were within 2% of the estimated averages. The errors are measured as percentage relative errors from the simulation estimate. To show the advantages in using ODE analysis for larger multiplicities we consider three scenarios: the first scenario uses the parameters as shown in Fig. 2 (we denote this by the label *x1*); the second scenario uses the same service demands, but all multiplicities for processors and tasks are increased by a factor 10 (label *x10*); the

	Scenario x1			*Scenario x10*			*Scenario x20*		
Metric/Kind/Name	ODE	Sim.	LQNS	ODE	Sim.	LQNS	ODE	Sim.	LQNS
Th / act / prepare	25.73	10.28	2.15	1.88	1.39	51.93	0.95	0.85	62.39
Th / entry / visit	25.78	10.65	2.11	1.86	1.50	51.94	0.98	0.79	62.38
Th / task / Server	25.79	10.46	2.10	1.85	1.51	51.94	0.97	0.88	62.38
Ut / proc / PClient	25.80	10.37	2.19	1.83	1.54	51.96	0.96	0.87	62.38
Ut / task / Server	5.45	6.30	1.49	1.68	0.69	11.04	1.67	0.82	14.70
Ut / task / Backup	26.00	10.59	2.17	1.72	1.24	52.10	1.17	1.21	62.38
RT / entry / think	20.42	9.29	2.35	1.80	1.52	108.13	0.95	0.86	165.85
RT / entry / save	65.42	48.93	11.44	6.24	5.63	503.64	0.87	0.84	787.90
RT / entry / notify	0.01	0.25	0.99	0.00	0.00	0.03	0.03	0.03	0.00
RT / entry / visit	37.20	4.70	70.61	19.05	1.20	214.83	17.89	2.71	321.06
Average	**25.76**	**12.18**	**9.76**	**3.79**	**1.62**	**109.76**	**2.65**	**0.99**	**160.14**

Table 2: Relative percentage errors of solvers against LQSim.

third scenario is doubled again in the same way (label *x20*). (For convenience, all scenarios are available for download as separate input files.)

The numerical results of DiffLQN are presented in Tab. 1. However, to reduce clutter only a selection of all performance estimates are presented. In particular, we removed repeated throughput estimates that were equal to those already found in the table. (This can happen when certain activities are performed sequentially, for instance prepare and display have the same steady-state throughput). The first column gives the type of the measure as a triple consisting of a metric — throughput (Th), utilization (Ut), processor utilization (PU), or response time (RT) — kind of LQN entity, and LQN entity name. The other columns show the performance estimates from ODE analysis and simulation in both scenarios, together with the percentage relative errors.

Overall, we can make the following main observations:

- Despite the low multiplicities of processors and tasks in scenario *x1*, the ODE estimates enjoy good accuracy in most cases.
- The highest errors in all scenarios occur for response-time metrics (entries save, visit). This confirms that response times can be challenging to approximate, because the errors of the basic metrics from which they are computed through Little's law can propagate [20].
- Scaling up multiplicities in scenarios *x10* and *x20* shows a considerable improvement on the accuracy, with the error dropping below 1% for all but one value, despite the fact that the model has populations of entities in the order of tens, which is significantly away from a limiting regime with infinitely many entities, where the ODE estimate is asymptotically exact.
- In scenario *x10*, the largest error is roughly halved. Although still large, it is possible to notice that the trend of that response-time metric is followed fairly well. All other errors drop to values well within the requested simulation accuracy and shrink even further in the *x20* scenario.
- It can be observed that the only error that maintains a value above 1% in the *x20* scenario coincides with the smallest value. We remark that this is an instance where the percentage relative error may not be very informative because it tends to penalize small variations in metrics that have small "true" values to start with [5].

	Runtime (mm:ss.ms)		
Solver	*x1*	*x10*	*x20*
LQSim	01:04.6	15:01.3	30:03.1
LQNS	00:00.1	00:00.1	00:00.1
Sim.	02:12.0	06:16.4	12:38.5
ODE	00:03.7	00:03.7	00:03.7

Table 3: Runtime comparison.

Table 2 shows how the performance of DiffLQN compares to LQNS in the running example. For this purpose, the ODE-based solver as well as the DiffLQN simulation (sim) and the analytical LQNS solver were measured against the result of the simulation tool included in LQNS (LQSim). To avoid clutter, this table has been reduced to only a few representative values, and shows only the relative errors, omitting the explicit results:

- In scenario *x1*, LQNS clearly provides the best approximation and the ODE solution shows the lowest accuracy, while the simulation on the PEPA model lies between the two. Despite the discrepancies, results of DiffLQN are in the correct order of magnitude and proportions are kept intact.
- As the size of the scenario is scaled up, the errors of the analytical LQNS solver rise dramatically, while the approximation by DiffLQN improves. In scenario *x10*, the accuracy of both DiffLQN results is already better than LQNS in the unscaled scenario.
- Interestingly, the most extreme errors in LQNS appear for those elements where DiffLQN has the lowest accuracy in the unscaled scenario. It seems that approximation difficulty for specific values is more dependent on the structure of the problem itself than on the solution method. The response time of save appears to be especially difficult to approximate, as could already be seen from Table 1.

The runtime comparison in Tab. 3 shows that LQNS is extremely fast in this example, with no noticeable runtime

increase in the scaled versions. The ODE-based solver is only slightly slower, and equally consistent. Runtimes of both simulations scale with the size of the model, but the runtime of the DiffLQN simulation rises at a slower rate.

6. CONCLUSION

This paper has presented DiffLQN, a tool that supports differential-equation analysis for layered queuing networks (LQNs). It is under active development and new features have been planned in order to increase its usability and applicability. Being based on Eclipse's Xtext framework, a natural evolution will be to provide a graphical user interface as an Eclipse plug-in, with the possibility of drawing LQNs in addition to specifying them textually. To enhance the capability of conducting large experiments such as what-if scenarios or capacity-planning studies, we plan to augment the syntax with parametric variables that can be instantiated (and the resulting model evaluated) over user-defined ranges. The numerical analysis of ODEs gives the time-course evolution of the queue-length process at each station, from which the steady-state LQN metrics are derived. Future releases will make these traces available to the user in order to obtain performance indices of the transient regime on an LQN as well.

From a more theoretical perspective, DiffLQN will allow us to carry out more extensive analyses of the error behavior of the differential analysis with respect to the ground truth of simulation as well as to alternative analytical techniques based on mean value analysis. Without the automated support offered by DiffLQN, these studies cannot but be performed manually on selected model instances, as has been done in the literature [17, 18, 19], necessarily limiting their scope of validity.

Acknowledgment

This work was partially supported by the EU project QUANTICOL, 600708.

7. REFERENCES

[1] U. M. Ascher and L. R. Petzold. *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations*. SIAM, 1988.

[2] S. Becker, H. Koziolek, and R. Reussner. Model-based performance prediction with the palladio component model. In *Proceedings of the 6th International Workshop on Software and Performance*, WOSP '07, pages 54–65, New York, NY, USA, 2007. ACM.

[3] Carleton University Software Performance Research Group. Layered queueing research resource page. http://www.layeredqueues.org/, Aug. 2014.

[4] K. M. Chandy and D. Neuse. Linearizer: A heuristic algorithm for queueing network models of computing systems. *Commun. ACM*, 25(2):126–134, 1982.

[5] D. L. Eager and J. N. Lipscomb. The AMVA priority approximation. *Perf. Eval.*, (8):173–193, 1988.

[6] G. Franks, T. Al-Omari, M. Woodside, O. Das, and S. Derisavi. Enhanced modeling and solution of layered queueing networks. *IEEE Trans. Software Eng.*, 35(2):148–161, 2009.

[7] D. Gillespie. Exact stochastic simulation of coupled chemical reactions. *Journal of Physical Chemistry*, 81(25):2340–2361, December 1977.

[8] J. Hillston. *A Compositional Approach to Performance Modelling*. Cambridge University Press, 1996.

[9] J. Hillston, M. Tribastone, and S. Gilmore. Stochastic process algebras: From individuals to populations. *The Computer Journal*, 2011.

[10] Institute of Systems Biology. BioUML documentation. http://www.biouml.org/index.shtml, May 2015.

[11] H. Koziolek and R. Reussner. A model transformation from the palladio component model to layered queueing networks. In *SIPEW*, 2008.

[12] T. G. Kurtz. Solutions of ordinary differential equations as limits of pure Markov processes. *J. Appl. Prob.*, 7(1):49–58, April 1970.

[13] K. R. Pattipati, M. M. Kostreva, and J. L. Teele. Approximate mean value analysis algorithms for queuing networks: Existence, uniqueness, and convergence results. *J. ACM*, 37(3):643–673, 1990.

[14] J. Perez and G. Casale. Assessing SLA compliance from palladio component models. In *2nd Workshop on Management of resources and services in Cloud and Sky computing (MICAS)*, pages 409–416, Sept 2013.

[15] J. A. Rolia and K. C. Sevcik. The method of layers. *IEEE Trans. Software Eng.*, 21(8):689–700, 1995.

[16] W. J. Stewart. *Probability, Markov Chains, Queues, and Simulation*. Princeton University Press, 2009.

[17] M. Tribastone. Relating layered queueing networks and process algebra models. In *WOSP*, 2010.

[18] M. Tribastone. Approximate mean value analysis of process algebra models. In *MASCOTS*, pages 369–378, 2011.

[19] M. Tribastone. A fluid model for layered queueing networks. *IEEE Trans. Software Eng.*, 39(6):744–756, 2013.

[20] M. Tribastone, J. Ding, S. Gilmore, and J. Hillston. Fluid rewards for a stochastic process algebra. *IEEE Trans. Software Eng.*, 38:861–874, 2012.

[21] M. Tribastone, A. Duguid, and S. Gilmore. The PEPA Eclipse Plug-in. *Performance Evaluation Review*, 36(4):28–33, March 2009.

[22] M. Tribastone and S. Gilmore. Automatic Extraction of PEPA Performance Models from UML Activity Diagrams Annotated with the MARTE Profile. In *WOSP*, 2008.

[23] M. Tribastone and S. Gilmore. Automatic Translation of UML Sequence Diagrams into PEPA Models. In *QEST*, 2008.

[24] M. Tribastone, S. Gilmore, and J. Hillston. Scalable differential analysis of process algebra models. *IEEE Trans. Software Eng.*, 38(1):205–219, 2012.

[25] M. Woodside, D. C. Petriu, D. B. Petriu, H. Shen, T. Israr, and J. Merseguer. Performance by Unified Model Analysis (PUMA). In *WOSP*, 2005.