Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 170 (2020) 498–505

Procedia Computer Science

www.elsevier.com/locate/procedia

The 11th International Conference on Ambient Systems, Networks and Technologies (ANT)
April 6 - 9, 2020, Warsaw, Poland

Exploiting Queuing Networks to Model and Assess the Performance of Self-Adaptive Software Systems: A Survey

Davide Arcelli[a,*]

[a]*Department of Information Engineering, Computer Science and Mathematics, University of L'Aquila, via Vetoio 1, L'Aquila, 67100, Italy*

Abstract

Self-adaptation has emerged as a primary concern in the context of modern software systems, due to the high dynamicity of the environments where they operate, which implies the need for such systems to properly face significant degrees of uncertainty. To this aim, much work has been done, mainly by coupling autonomic managers to the managed subsystem which perceives and affects the environment through its sensors and actuators, respectively. Such coupling often results into MAPE-K feedback loop(s), i.e. a Knowledge (K)-based architectural model that divides the adaptation process into four activities, namely Monitor (M), Analyze (A), Plan (P) and Execute (E). Performance modeling notations, analysis methods and tools, have been exploited and coupled to other kinds of techniques (e.g. control theory, machine learning) for modeling and assessing the performance of autonomic managers, possibly aimed at supporting the identification of more convenient architectural alternatives. Since moving in such a big arena is not trivial and it is easy to be overwhelmed, in this literature survey, we focus on a particular performance modeling paradigm, namely Queuing Networks, with the aim of clarifying the state-of-art in exploiting such a notation to model and assess performance of Self-Adaptive Software Systems. We conclude by bringing out some research opportunities that may be worth exploring in the near future.

Keywords: Self-Adaptive Software Systems; Software Architectures; Autonomous Systems; Performance Engineering; Queuing Networks.

1. Introduction

Self-adaptation has emerged as a primary concern in the context of modern software systems, due to the high dynamicity of the environments where they operate, which implies the need for such systems to properly face significant degrees of uncertainty [27, 12].

* Corresponding author.
E-mail address: davide.arcelli@univaq.it

1877-0509
10.1016/j.procs.2020.03.108

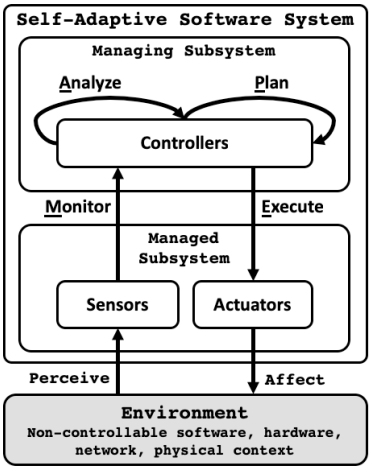

Fig. 1. Reference model for self-adaptation.

To this aim, much work has been done, mainly by introducing autonomic managers – namely *managing subsystems* – comprising controllers implementing the adaptation logic; The latter are coupled with *managed subsystems*, which comprise the application logic providing system's domain functionalities while perceiving and affecting the environment by means of sensors and actuators, respectively [31]. Such coupling often results into *MAPE-K feedback loop(s)* [21], i.e. a Knowledge (K)-based architectural model that divides the process of adaptation into four phases: Monitor (M), Analyze (A), Plan (P), and Execute (E), as illustrated in the typical reference model for self-adaptation illustrated in Figure 1.

In this domain, performance represents a top concern, as highlighted by several literature reviews that have been conducted in the last decade [31, 30, 7] in the macro-area of Self-Adaptive Software Systems (SASSs), with the aim of classifying the different aspects involved in such a context. In fact, performance modeling notations, analysis methods and tools, have been introduced and coupled to techniques of different nature, in order to support performance modeling and assessing of managing subsystems. For example, Control Theory – i.e. a mathematical approach to properly control continuously operating dynamical systems – allows to introduce (global or local) MAPE-K feedback controllers able to provide formal guarantees within performance models such as Queuing Networks (QNs) [3, 4, 18]; Machine Learning – i.e. an approach that builds mathematical models which allow to make predictions or decisions without being explicitly programmed to perform the task – and Model-Driven Engineering (MDE) – i.e. a paradigm that is massively based on (e.g. architectural, analysis) models and their manipulation (by means of, e.g., refactoring, transformation), in order to introduce automation into the system development process – can be used in heuristics or semi-formal approaches to (semi-)automatically reason and decide about adaptation, driven by performance requirements [20, 13, 5, 17, 6, 11, 24, 22]. However, the main goal of existing approaches is to introduce self-adaptation mechanisms and validate their performance once enabled. While the mathematical underpinning and soundness of those techniques are assumed for granted but usually not provided, their different natures and the specificity of the introduced adaptation mechanisms bring to a global under-exploitation of methods and tools in the field, as highlighted in a study by Weyns et al. [31].

The huge number of dimensions over which such a big arena spans, makes the domain investigation very hard and subject to entropy. For this reason, we focus onto a particular area, i.e. performance modeling and assessment. More particularly, with respect to the wide plethora of performance notations (QNs, Petri-nets, Markov-chains, etc.), we restrict the focus to a specific paradigm, i.e. QNs, aimed at clarifying the state-of-art with respect to approaches dealing with performance modeling and assessment of SASSs by means of this latter paradigm.

The paper is structured as follows: Section 2 devises the knowledge base carrying out the set of surveyed approaches (Section 2.1) and illustrates the classification scheme (Section 2.2). On these bases, our findings are provided in Section 3, together with a discussion of potential strengths and limitations of the surveyed approaches modulo the classifying dimensions (Sections 3.1 and 3.2, respectively). Section 4 concludes the paper by bringing out some research opportunities that may be worth exploring in the near future.

2. Methodology

2.1. Knowledge base

This survey grounds on a knowledge base selected among a number of literature studies addressing performance concerns, while spanning over several other dimensions. In particular, we consider two surveys, one by Weyns et al. [31] and one by Becker et al. [7], which reported the state-of-art on addressing non-functional concerns by means of formal notations and model-driven engineering, respectively, until 2012. While doing this, we also take into account possible evolutions/extensions of the considered approaches that might have introduced new features or improved the existing ones. Additionally, we consider a more recent systematic study by Shevtsov et al. [30], which reviewed

the literature with respect to approaches exploiting Control Theory to introduce self-adaptation and providing formal guarantees of non-functional properties.

Among the approaches included in those three studies, five exploit the QN paradigm to address performance modeling and analysis, namely SimuLizar [6], QoSMOS [14, 11], SAFCA [32, 24], ICAC [20][1] and Adaptive Queuing Networks (AQNs) [3, 4]. SimuLizar [6] extends Palladio's modeling approach [9] with a self-adaptation viewpoint and a simulation engine, enabling performance prediction of SASSs and allowing the analysis of transient adaptation phases. QoSMOS [14, 11] is a service-based framework capable of dynamically selecting services to face run-time changes, by choosing among a set of functionally equivalent services for each component of a service-based system, driven by performance and reliability requirements. SAFCA [32, 24] enables self-adaptation in the context of distributed and concurrent software architectures, by changing demands for software performance improvement and/or reliability enhancement. ICAC [20] exploits decision-tree learners to produce rules sets representing adaptation policies for (Layered) QNs. AQNs [3, 4] represent a QN specialization which exploits Control Theory to enable self-adaptation of QN stations over a set of predefined service rates, in order to maintain a certain queue length for each station.

Furthermore, by additional search performed on Google Scholar[2] and Scopus[3], two more recent approaches can be identified: the first one is by Incerto et al. [18] and exploits Efficient Model Predictive Control (EMPC) to enable performance-driven self-adaptation within QNs[4]; the second one is by Arcelli et al. [1] and introduces the so-called SMAPEA QNs exploiting advanced QN constructs such as class-switches in order to enable mode-based adaptation.

All the approaches exploit QNs as analytical models to be solved in order to obtain performance indices of interest driving self-adaptation. However, self-adaptation is introduced at level of system architecture – grounding on a different modeling notation – by SimuLizar [6] and QoSMOS [14, 11], whilst the other approaches introduce self-adaptation directly into the QNs, as detailed in Section 3.1.

2.2. *Classification scheme*

In order to compare the contributions of the considered approaches based on some dimensions of interest, we take inspiration from the different classification schemes introduced by the three systematic studies in the knowledge base, i.e. [31], [7] and [30]. We thus devise the following classification scheme:

Foundations: This category describes the ***foundational paradigms*** of the approaches. For our aims, all the considered approaches ground on QNs. Besides, examples of foundational paradigms include MDE, Machine Learning and Control Theory.

Time of application: This category distinguishes between approaches applicable at ***design-time*** and/or ***run-time***. Approaches belonging to the former category may be exploited, e.g., to identify proper adaptation strategies; whereas, approaches from the latter category may, e.g., "measure the environment" aiming at predicting system's performance trend.

Adaptation: With this category we characterize the approaches based on the following self-adaptation concerns:

***Proactive or reactive*:** "We call an adaptation strategy *reactive*, if the system triggers its self-adaptation when a goal is already violated. If the system predicts that it might miss a goal some time in the near future and hence adapts itself preventively, we call that *proactive*" [7].

***Type*:** Based on the considered approaches and taking inspiration from [30], we distinguish among three different types of adaptation:

- *Component adaptation* "refers to changes at the level of software components, such as the load of services and the degree of parallelism that components process requests" [30].

[1] Since the approach in [20] has not been named, we introduced the acronym ICAC from its publication venue, i.e. the International Conference on Autonomic Computing.

[2] https://scholar.google.it/, last accessed on 10th February, 2020.

[3] https://www.scopus.com/, last accessed on 10th February, 2020.

[4] Since the approach in [18] has not be named, we introduced the acronym EMPC from its main control technique publication venue, i.e. Efficient Model Predictive Control.

- *Parametric adaptation* "refers to changing the values of variables of the application software or middleware services. These types of actuators are typically domain-specific; examples are the degree of video compression and the length of a queue with pending requests that need to be processed" [30].
- *Mode adaptation* "refers to a variation in the mode of operation, which can be either *mode change* or *mode switch*[5]. An example of a mode change is an increment in the quality of content that is being served by a video application; an example of mode switch is an alteration of the buffering schema of a video application" [30].
- *Architecture reconfiguration* "refers to a run-time adaptation of the architectural structure or behavior of the application" [30], which basically means selecting an (optimal) alternative system architecture and actually re-arranging the current one conforming to implement the former.

***Mechanisms*:** Adaptation mechanisms can be *introduced* or *assumed* by the approaches. In the former case, the approaches introduce `MAPE-K` feedback loops in order to Monitor software systems and consequently Execute behavioral adaptation after Analysis and Planning. Hence, the software system "is made self-adaptive" through the proposed contribution. In the latter case, an approach may assume the system is self-adaptive by itself, hence the focus shifts on to its non-functional modeling and analysis.

Architecture: In this category, architectural concerns are devised, that are:

***Architectural Paradigm*:** The considered approaches can be based on *components*, *services* (Service-Oriented Architectures – *SOA*), *concurrent* architectures or *multi-tier* applications[6].

***Architectural Model*:** In case of approaches exploiting MDE, modeling languages such as *UML*, *BPEL*, as well as Domain-Specific Languages (*DSLs*), are used to model the system architecture.

Performance Analysis: In accordance with [7], this classification criterion characterizes the approaches according to the adopted analysis method and models, possible transformations provided by the approach (e.g. from architectural system model to performance model) and if additional models are exploited (e.g. pivotal intermediate system abstractions).

***Method*:** Two performance analysis methods are considered by the examined approaches, i.e. *analytic* and *simulation* [19].

***Analysis Model*:** As QNs represent the main foundational paradigm of our approach, we restrict to approaches exploiting QNs [23] or a particular extension of the latter, namely *Layered Queuing Networks* (LQNs) [15], as performance analysis model.

***Transformation*:** Approaches grounding on MDE paradigm, analysis models are derived – automatically or semi-automatically – by architectural models via model transformations, whose provisioning represent a classification criterion.

***Additional Models*:** We also report on whether pivotal representations of the system are exploited for performance analysis aims.

Applicability: Taking inspiration from [7], the applicability depends on to which extent the approach provides tool support and it can be shown by a proof-of-concept implementation, or a case study.

***Analysis Tools*:** This represents whether the performance analysis is tool-supported.

***MDE Tools*:** Approaches grounding on the MDE paradigm may provide tool support for the involved MDE activities, e.g. model transformation from architectural to performance model(s).

***Proof-of-concept*:** The approach validity is shown based on an ad-hoc system model or implementation.

***Case Study*:** The approach validity is illustrated based on a real(istic) system design or implementation.

[5] The concept of system *mode* is known since more that a decade in the context of dynamic adaptive systems and has been used to devise different run-time configurations among which the system may transit for self-adaptation [25]. This definition makes mode adaptation a particular case of architecture reconfiguration.

[6] Note that the latter two options are not mutually exclusive with the others (e.g. concurrent architectures by component-based modeling); however, this does not emerge explicitly from the considered approaches.

3. Results

Table 1 classifies the considered approaches based on the classification scheme devised in Section 2.2.

Table 1. Classification of the surveyed approaches.

	SimuLizar	QoSMOS	SAFCA	ICAC	AQNs	EMPC	SMAPEA QNs
References	[6]	[14], [11]	[32], [24]	[20]	[3], [4]	[18]	[1]
Systematic studies	[7]	[7], [31]	[7]	[31]	[30]	-	-
Foundations							
Foundational Paradigms	QNs, MDE	QNs, MDE	QNs, MDE	QNs, Machine Learning	QNs, Control Theory	QNs, Control Theory	QNs
Time of application							
Design-/Run-time	design-time	run-time	run-time	design-time	design-time	run-time	both
Adaptation							
Pro-/Reactive	reactive	proactive	reactive	reactive	reactive	proactive	pro-, reactive
Type	architecture reconfiguration	architecture reconfiguration	architecture reconfiguration	architecture reconfiguration	mode change	comp./par. adaptation	mode switch
Mechanisms	introduced	introduced	introduced	introduced	introduced	introduced	assumed
Architecture							
Architectural paradigm	components	SOA	concurrent	multi-tier	components	components	components
Architectural model	PCM	BPEL	(unspecified)	-	-	-	-
Performance analysis							
Method	simulative	analytical	analytical	analytical	simulative	analytical	simulative
Analysis model	QN	QN	LQN	LQN	QN	QN	QN
Transformation	yes	-	-	-	-	yes	-
Additional models	-	-	-	-	-	yes	-
Applicability							
Analysis Tools	yes	yes	yes	yes	yes	yes	yes
MDE Tools	-	yes	-	-	-	-	-
Proof-of-Concept	yes	yes	yes	-	yes	-	yes
Case Study	prototypal	-	-	yes	-	yes	-

3.1. Findings

A very important characteristic that distinguishes SMAPEA QNs from other approaches is the fact that the latter are aimed at introducing adaptation mechanisms (i.e. MAPE-K feedback loops), whilst the former assume such mechanisms are implemented by the system (i.e. the system is self-adaptive already). In other terms, the "input" taken by related approaches is a managed system which is made self-adaptive by introducing a managing system that controls it; in [1], instead, the input is composed by both the managed and managing systems, assuming the latter being in charge for self-adaptation. As a consequence, since pro- and re-activeness are characteristics owned by managing systems[7] and the latter is part of the input, SMAPEA QNs could be applied to both proactive and reactive self-adaptation. As a further consequence, the work in [1] has the potential for being adopted to model and assess the performance of other approaches, in a kind of "meta-application" whose input is the SASS resulting from the union between the managed system and the corresponding managing system introduced by the other approach.

In most of other approaches, i.e. SimuLizar, QoSMOS, SAFCA and ICAC, self-adaptation is introduced by enabling architecture reconfiguration that, in case of SimuLizar and QoSMOS, takes place onto architectural models rather than performance ones, due to the exploitation of the MDE paradigm, which implies a need for model transformation towards the QN notation; in the other two approaches adopting architecture reconfiguration, i.e. SAFCA and ICAC, the latter takes place directly onto the QNs, without involving any additional architectural model[8]. Both SAFCA and ICAC rely on LQNs [15], i.e. a well-known extension of QNs, as performance analysis notation, which more likely resulted to be suitable to model the addressed architectural paradigms, i.e. concurrent and multi-tier systems, respectively. Differently from the four aforementioned existing approaches, EMPC [18] and AQNs [3, 4] – that rely on control theory – introduce self-adaptation by automatically tuning pre-selected knobs, aimed at satisfying global [18] or local [3, 4] performance requirements. In particular, EMPC allows software engineers to select routing probabilities and concurrency level (parameter adaptation), as well as service rates (component adaptation), as knobs;

[7] In particular, they concern analysis and planning phases of the MAPE-K feedback loop.
[8] Readers interested in a comparison between working at software and performance model sides may refer to [2].

then, a Model Predictive Control (MPC) formulation is automatically derived, "suitable to continuously configure the selected knobs and track the desired performance requirements" (e.g. mean system throughput or response time), in a proactive manner[9]. In AQNs, instead, local controllers are properly synthesized and coupled with QN stations in order to control their queue lengths by adapting, in a reactive manner, over a (discrete) set of predefined service rates that abstract the provision of different service quality levels (e.g. gold, silver, bronze users), thus resulting into a *mode-change* adaptation type.

Like AQNs, SMAPEA QNs address mode adaptation, but they rely on *mode-switching*, which can be suitably modeled by exploiting the *class-switch* QN construct. The latter is able to transform jobs based on the classes they belong to, conforming to predefined probabilities. For instance, $Class_1$ jobs are transformed into $Class_2$ and $Class_3$ jobs with probability p and $1 - p$, respectively. Mode-specific job classes can be thus devised, which implies the specification of corresponding switching probabilities. To get the intuition of what these probabilities represent, an effective example is represented by the case study presented in the corresponding paper [1], i.e. a SASS for emergency handling that may work either in *normal* or *critical* mode: the latter is adopted in case an emergency occurs (e.g. a fire alarm) and evacuation of a building (e.g. a museum) is needed. Such probabilities are thus fundamental domain-specific input parameters to be provided, as for any stochastic performance model, representing a part of the system's operational profile, often referred as *mode profile* [26]. To this aim, offline estimation techniques, as well as online system (prototype) monitoring, might be exploited, possibly using machine-learning. That said, SMAPEA QNs can be used both at design- and run-time (offline and online, respectively), based on available information that strictly depends on the current phase of the system's life-cycle.

Performance analysis methods of the considered QN-based approaches equally distributes between analytical and simulative. The latter method is adopted when the produced QN models cannot be solved analytically [10] as, e.g., in case of SMAPEA QNs, which contain fork/join and class-switches.

All the considered approaches rely on available analysis/simulation tools, except SAFCA. In particular: JMT [10] is used by QoSMOS and SMAPEA QNs; SimuCom [9] and ProtoCom [8] are used by SimuLizar; ICAC relies on LQNS [15]; Modelica [16] is used by AQNs; finally, EMPC uses the well-known CPLEX tool.

Most of the approaches provide a proof-of-concept implementation, i.e. SimuLizar in its initial version, QoSMOS, SAFCA and AQNs; ICAC, EMPC and the latest SimuLizar [6], have been applied to real(istic) case studies, that is the same approach used for validating SMAPEA QNs.

3.2. Discussion

From a notational point of view, exploiting the MDE paradigm allows to deal with more understandable system designs, but very likely introducing a need of model transformation(s) from/to QNs and/or additional models resulting into an overhead. Such overhead may significantly increase in case support at both design- and run-time is introduced, due to the fact that architectural and/or performance models shall be automatically generated at run-time.

Furthermore, although the analytical resolution of QN models may fasten the whole process, simulation might be needed, e.g., when QNs are not in product-form [23] or when advanced constructs such as class-switches are exploited. This would very likely introduce significant additional overhead, due to the fact that simulation shall be executed several times in order to be as confident as possible about the obtained results.

Control Theory and/or Machine Learning may help in making the system more autonomous, both in case of proactive and reactive adaptation. Despite from this, existing approaches tend to support one adaptation type (mostly architectural reconfiguration), representing the one onto which the approach is tailored.

Assuming that the system is self-adaptive, i.e. involving the managed subsystem in the performance modeling and the subsequent analyses, is an advancement to take into consideration, since the managed subsystem represents the part of the system interacting with the environment, hence it is involved in dynamics that can heavily affect performance.

The availability of analysis tools is crucial for the adoption of any approach for self-adaptation, because developing ad-hoc analysis tools can be very costly. Hence, relying on existing widespread and well-assessed tools such as [10] represents a valuable choice. Similar arguments apply to system modeling. In fact, while exploiting the MDE

[9] The MPC formulation is an additional textual model obtained by means of a Model-to-Text transformation. For this reason, in Table 1 we put *yes* into "Transformation" and "Additional Models" classification criterion.

paradigm, well-assessed modeling notations that allow to embed performance parameters (e.g. workload, demands, indices), such as the Unified Modeling Language (UML) [28] with MARTE profile [29], shall be preferred to domain-specific languages. However, the latter might not be avoided in certain contexts, thus introducing the need of developing additional tools for, e.g., visual editing and model validation.

Finally, although proof-of-concepts are useful for preliminary validation of approaches for SASSs, case studies coming from the real world, possibly involving a huge number of components, should be addressed. Mathematical evidence of approach correctness and effectiveness shall be provided in case control theory is exploited, representing an added-value in general terms (i.e. even in case control theory would not be put in place).

4. Conclusion

In this paper, we have surveyed the literature with respect to approaches dealing with performance modeling and assessment of self-adaptive software systems by means of the Queuing Network paradigm.

Results have shown that such paradigm can be successfully adopted as performance analysis model in the context of self-adaptive software systems. Different mechanisms for self-adaptation can be introduced by applying formal techniques and heuristics relying on Machine Learning or Control Theory, to the analysis or architectural models. In this latter cases, advanced Model-Driven Engineering techniques such as model transformation may be needed in order to switch from architectural to analysis perspective and vice-versa.

It is worth to notice that, in the current state-of-art concerning performance modeling and assessment of self-adaptive software systems by means of the Queuing Network paradigm, meta-heuristics have not been exploited yet. The latter could provide automated support performance-driven decision-making towards convenient alternative self-adaptive system architectures, by design-space exploration performed with, e.g., a genetic algorithm.

Mechanisms for self-adaptation are usually enabled by introducing autonomic manager(s) coupled to the parts of the system that provide system's domain functionalities, i.e. sensors and actuators. Since these parts of the system are involved in dynamics that can heavily affect performance, the scope of performance modeling and analysis has started to be widened, very recently, in such a way that sensors and actuators are part of system abstraction and, consequently, they are considered during performance analysis. Investigating in this sense represents an interesting research opportunity for the near future.

References

[1] Arbib, C., Arcelli, D., Dugdale, J., Moghaddam, M.T., Muccini, H., 2019. Real-time Emergency Response through Performant IoT Architectures, in: International Conference on Information Systems for Crisis Response and Management (ISCRAM), Valencia, Spain. pp. 388–401. URL: https://hal.archives-ouvertes.fr/hal-02091586.

[2] Arcelli, D., Cortellessa, V., 2013. Software model refactoring based on performance analysis: better working on software or performance side?, in: Buhnova, B., Happe, L., Kofron, J. (Eds.), Proceedings 10th International Workshop on Formal Engineering Approaches to Software Components and Architectures, FESCA 2013, Rome, Italy, March 23, 2013, pp. 33–47. URL: https://doi.org/10.4204/EPTCS.108.3, doi:10.4204/EPTCS.108.3.

[3] Arcelli, D., Cortellessa, V., Filieri, A., Leva, A., 2015. Control theory for model-based performance-driven software adaptation, in: Kruchten, P., Ozkaya, I., Koziolek, H. (Eds.), Proceedings of the 11th International ACM SIGSOFT Conference on Quality of Software Architectures, QoSA'15 (part of CompArch 2015), Montreal, QC, Canada, May 4-8, 2015, ACM. pp. 11–20. URL: https://doi.org/10.1145/2737182.2737187, doi:10.1145/2737182.2737187.

[4] Arcelli, D., Cortellessa, V., Leva, A., 2016. A library of modeling components for adaptive queuing networks, in: Fiems, D., Paolieri, M., Platis, A.N. (Eds.), Computer Performance Engineering - 13th European Workshop, EPEW 2016, Chios, Greece, October 5-7, 2016, Proceedings, Springer. pp. 204–219. URL: https://doi.org/10.1007/978-3-319-46433-6_14, doi:10.1007/978-3-319-46433-6_14.

[5] Barati, S., Bartha, F.A., Biswas, S., Cartwright, R., Duracz, A., Fussell, D.S., Hoffmann, H., Imes, C., Miller, J.E., Mishra, N., Arvind, Nguyen, D., Palem, K.V., Pei, Y., Pingali, K., Sai, R., Wright, A., Yang, Y.H., Zhang, S., 2019. Proteus: Language and runtime support for self-adaptive software development. IEEE Software 36, 73–82. URL: https://doi.org/10.1109/MS.2018.2884864, doi:10.1109/MS.2018.2884864.

[6] Becker, M., Becker, S., Meyer, J., 2013. Simulizar: Design-time modeling and performance analysis of self-adaptive systems, in: Kowalewski, S., Rumpe, B. (Eds.), Software Engineering 2013: Fachtagung des GI-Fachbereichs Softwaretechnik, 26. Februar - 2. März 2013 in Aachen, GI. pp. 71–84. URL: https://dl.gi.de/20.500.12116/17731.

[7] Becker, M., Luckey, M., Becker, S., 2012. Model-driven performance engineering of self-adaptive systems: A survey, in: Proceedings of the 8th International ACM SIGSOFT Conference on Quality of Software Architectures, Association for Computing Machinery, New York, NY, USA. p. 117–122. URL: https://doi.org/10.1145/2304696.2304716, doi:10.1145/2304696.2304716.

[8] Becker, S., Dencker, T., Happe, J., 2008. Model-driven generation of performance prototypes, in: Kounev, S., Gorton, I., Sachs, K. (Eds.), Performance Evaluation: Metrics, Models and Benchmarks, Springer Berlin Heidelberg. pp. 79–98.

[9] Becker, S., Koziolek, H., Reussner, R.H., 2009. The palladio component model for model-driven performance prediction. Journal of Systems and Software 82, 3–22. URL: https://doi.org/10.1016/j.jss.2008.03.066, doi:10.1016/j.jss.2008.03.066.

[10] Bertoli, M., Casale, G., Serazzi, G., 2009. Jmt: performance engineering tools for system modeling. SIGMETRICS Performance Evaluation Review 36, 10–15. doi:http://doi.acm.org/10.1145/1530873.1530877.

[11] Calinescu, R., Grunske, L., Kwiatkowska, M., Mirandola, R., Tamburrelli, G., 2011. Dynamic qos management and optimization in service-based systems. IEEE Transactions on Software Engineering 37, 387–409. doi:10.1109/TSE.2010.92.

[12] Cámara, J., Garlan, D., Kang, W.G., Peng, W., Schmerl, B.R., 2017. Uncertainty in Self-Adaptive Systems Categories, Management, and Perspectives. Technical Report. Institute for Software Research, Carnegie Mellon University.

[13] Elkhodary, A., Esfahani, N., Malek, S., 2010. Fusion: A framework for engineering self-tuning self-adaptive software systems, in: Proceedings of the Eighteenth ACM SIGSOFT International Symposium on Foundations of Software Engineering, Association for Computing Machinery, New York, NY, USA. p. 7–16. URL: https://doi.org/10.1145/1882291.1882296, doi:10.1145/1882291.1882296.

[14] Epifani, I., Ghezzi, C., Mirandola, R., Tamburrelli, G., 2009. Model evolution by run-time parameter adaptation, in: Proceedings of the 31st International Conference on Software Engineering, IEEE Computer Society, USA. p. 111–121. URL: https://doi.org/10.1109/ICSE.2009.5070513, doi:10.1109/ICSE.2009.5070513.

[15] Franks, G., Majumdar, S., Neilson, Petriu, D., Rolia, J., Woodside, M., 1996. Performance analysis of distributed server systems, in: In Proceedings of the 6th International Conference on Software Quality, pp. 15–26.

[16] Fritzson, P., Engelson, V., 1998. Modelica — a unified object-oriented language for system modeling and simulation, in: Jul, E. (Ed.), In Proceedings of the 12th European Conference on Object-Oriented Programming (ECOOP), Springer Berlin Heidelberg. pp. 67–90.

[17] Grassi, V., Mirandola, R., Randazzo, E., 2009. Model-driven assessment of qos-aware self-adaptation, in: Cheng, B.H.C., de Lemos, R., Giese, H., Inverardi, P., Magee, J. (Eds.), Software Engineering for Self-Adaptive Systems. Springer Berlin Heidelberg, pp. 201–222. URL: https://doi.org/10.1007/978-3-642-02161-9_11, doi:10.1007/978-3-642-02161-9_11.

[18] Incerto, E., Tribastone, M., Trubiani, C., 2017. Software performance self-adaptation through efficient model predictive control, in: Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering, IEEE Press. p. 485–496. doi:10.1109/ASE.2017.8115660.

[19] Jain, R., 1991. The art of computer systems performance analysis - techniques for experimental design, measurement, simulation, and modeling. Wiley professional computing, Wiley.

[20] Jung, G., Joshi, K.R., Hiltunen, M.A., Schlichting, R.D., Pu, C., 2008. Generating adaptation policies for multi-tier applications in consolidated server environments, in: Strassner, J., Dobson, S.A., Fortes, J.A.B., Goswami, K.K. (Eds.), 2008 International Conference on Autonomic Computing, ICAC 2008, June 2-6, 2008, Chicago, Illinois, USA, IEEE Computer Society. pp. 23–32. URL: https://doi.org/10.1109/ICAC.2008.21, doi:10.1109/ICAC.2008.21.

[21] Kephart, J.O., Chess, D.M., 2003. The vision of autonomic computing. IEEE Computer 36, 41–50. URL: https://doi.org/10.1109/MC.2003.1160055, doi:10.1109/MC.2003.1160055.

[22] Kounev, S., Brosig, F., Huber, N., Reussner, R.H., 2010. Towards self-aware performance and resource management in modern service-oriented systems, in: 2010 IEEE International Conference on Services Computing, SCC 2010, Miami, Florida, USA, July 5-10, 2010, IEEE Computer Society. pp. 621–624. URL: https://doi.org/10.1109/SCC.2010.94, doi:10.1109/SCC.2010.94.

[23] Lazowska, E.D., Zahorjan, J., Graham, G.S., Sevcik, K.C., 1984. Quantitative system performance - computer system analysis using queueing network models. Prentice Hall. URL: http://dl.acm.org/citation.cfm?id=2971.

[24] Lung, C., Zhang, X., Rajeswaran, P., 2016. Improving software performance and reliability in a distributed and concurrent environment with an architecture-based self-adaptive framework. Journal of Systems and Software 121, 311–328. URL: https://doi.org/10.1016/j.jss.2016.06.102, doi:10.1016/j.jss.2016.06.102.

[25] Morin, B., Barais, O., Nain, G., Jézéquel, J., 2009. Taming dynamically adaptive systems using models and aspects, in: 31st International Conference on Software Engineering, ICSE 2009, May 16-24, 2009, Vancouver, Canada, Proceedings, IEEE. pp. 122–132. URL: https://doi.org/10.1109/ICSE.2009.5070514, doi:10.1109/ICSE.2009.5070514.

[26] Musa, J.D., 1993. Operational profiles in software-reliability engineering. IEEE Software 10, 14–32. URL: https://doi.org/10.1109/52.199724, doi:10.1109/52.199724.

[27] Perez-Palacin, D., Mirandola, R., 2014. Uncertainties in the modeling of self-adaptive systems: a taxonomy and an example of availability evaluation, in: Lange, K., Murphy, J., Binder, W., Merseguer, J. (Eds.), ACM/SPEC International Conference on Performance Engineering, ICPE'14, Dublin, Ireland, March 22-26, 2014, ACM. pp. 3–14. URL: https://doi.org/10.1145/2568088.2568095, doi:10.1145/2568088.2568095.

[28] Rumbaugh, J., Jacobson, I., Booch, G., 2004. The Unified Modeling Language Reference Manual (2nd Edition). Pearson Higher Education.

[29] Selic, B., Grard, S., 2013. Modeling and Analysis of Real-Time and Embedded Systems with UML and MARTE: Developing Cyber-Physical Systems. 1st ed., Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[30] Shevtsov, S., Berekmeri, M., Weyns, D., Maggio, M., 2018. Control-theoretical software adaptation: A systematic literature review. IEEE Trans. Software Eng. 44, 784–810. URL: https://doi.org/10.1109/TSE.2017.2704579, doi:10.1109/TSE.2017.2704579.

[31] Weyns, D., Iftikhar, M.U., de la Iglesia, D.G., Ahmad, T., 2012. A survey of formal methods in self-adaptive systems, in: Desai, B.C., Vassev, E., Mudur, S.P. (Eds.), Fifth International C* Conference on Computer Science & Software Engineering, C3S2E '12, Montreal, QC, Canada, June 27-29, 2012, ACM. pp. 67–79. URL: https://doi.org/10.1145/2347583.2347592, doi:10.1145/2347583.2347592.

[32] Zhang, X., Lung, C., Franks, G., 2010. Towards architecture-based autonomic software performance engineering, in: Drira, K. (Ed.), 4e Conférence francophone sur les Architectures Logicielles, CAL 2010, Pau, France, 9-11 mars 2010, Cépaduès-Éditions. pp. 144–156. URL: http://editions-rnti.fr/?inprocid=1000909.