Journal of Ubiquitous Systems & Pervasive Networks
Volume 14, No. 2 (2021) pp.27 -35

Understanding and Comparing Approaches for Performance Engineering of Self-adaptive Systems Based on Queuing Networks

Davide Arcelli[a] *

[a]*Università degli Studi dell'Aquila, L'Aquila, Italy, 67100*

Abstract

Enabling self-adaptation within hardware/software systems is a complex task, mainly due to environment uncertainty that has to be faced while the system is providing its functionalities. Besides, non-functional goals that have to be met by the system may be introduced, defining Quality-of-Service (QoS) requirements which drive the adaptation.

This paper enhances a previous study which surveyed the literature with respect to performance-driven self-adaptation, supported by the Queuing Network paradigm. The seven approaches identified in previous work are detailed in this paper based on a well-defined taxonomy deriving from the former's classification scheme and spanning over different dimensions, with particular emphasis on the way adaptation mechanisms are introduced, e.g. available knobs, non-functional goals, sources of uncertainty. Based on such taxonomy, internal characteristics of those approaches are described, as well as commonalities and differences, aimed at providing a detailed view of the current state-of-art in the context of performance-driven self-adaptation supported by the Queuing Network paradigm.

Keywords: *Self-Adaptive Systems, Software Architecture, Autonomous Systems; Software Performance Engineering; Queuing Networks.*

1. Introduction

Recent advancements in IT technologies have brought to a wide plethora of application domains for modern hardware/software systems. Many of those envision the latter operating in dynamic environments with different sources of uncertainty that they have to face while providing their functionalities [1, 2].
To this aim, system architectures have switched to a more "elastic" paradigm, which allowed to develop the so-called Self-adaptive Systems (SaSs) [3].
A SaS is composed by a *managed* and a *managing subsystem*: the former comprises sensing and actuating components which allow to perceive and affect the environment, respectively; the latter subsystem, instead, includes controllers that exploit sensed data in order to devise adaptation of system's behavior resulting into actuation.
Hence, the two subsystems are coupled each other and such coupling often results into `MAPE-K` *feedback loop(s)* [4], 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 of Fig. 1.

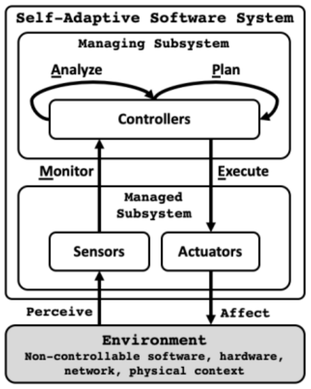

Fig. 1. Reference model for self-adaptation.

In this domain, non-functional goals have been taken into account by many approaches. In particular, performance has emerged as a top concern, as highlighted by several literature reviews that have been conducted in the last decade [3, 5, 6].
Modeling and analysis notations have been introduced in order to represent SaSs and assess their performance. Besides, several techniques have been exploited in order to optimize non-functional attributes of such systems.

* Corresponding author.
E-mail: davide.arcelli@univaq.it, davide.arcelli@gmail.com

DOI: 10.5383/JUSPN.14.02.004

For example, Control Theory (CT) [7] – i.e. a mathematical approach to properly control continuously operating dynamical systems – allows to introduce (global or local) `MAPE-K` feedback controllers within Queuing Networks (QNs) [8], aimed at providing formal performance guarantees [9, 10, 11]; As further examples, Machine Learning (ML) [12] – i.e. an approach that builds mathematical models which allow to make predictions or decisions without being explicitly programmed to perform the task – and Search-space Exploration [13] – i.e. metaheuristics looking for near-optimal solutions to optimization problems – can be used to automatically reason and decide about adaptation, driven by performance requirements [14, 15, 16, 17, 18, 19, 20, 21, 22].

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, in a recent survey [23] I have focused onto a particular non-functional aspect of SaSs, i.e. performance. More in detail, in the wide plethora of available performance notations – e.g. QNs [8, 24], Markov models [25], Petri-nets [26], etc. – I have restricted such literature study to QNs, which represent one of the most addressed performance modeling and analysis paradigms [3, 5, 6].

Seven approaches enabling performance-driven self-adaptation of SaSs by exploiting QNs have been identified through previous investigation, whose main characteristics have been preliminarily pointed out and discussed.

This paper enhances the previously published survey [23] by going into details of the adaptation mechanisms introduced by those approaches. First, a well-defined taxonomy is presented, enhancing the classification scheme of the previous survey. Then, the considered approaches are characterized based on the presented taxonomy and such characterization is finally used to detail the approaches and highlight commonalities and differences, in the light of their latest advancements.

As a result, this work contributes with the previous one to provide a detailed view of the current state-of-art in the context of performance-driven self-adaptation supported by the QN paradigm.

The paper is structured as follows: Sec. 2 describes the knowledge base including of surveyed approaches (Sec. 2.1) and presents a taxonomy for their classification (Sec. 2.2). On these basis, Sec. 3 provides the classification of the considered approaches and then describes and compares them (Secc. 3.1 and 3.2, respectively). Results are summarized in Sec. 4, whilst Sec. 5 concludes the paper.

2. Methodology

2.1. Identifying the Approaches for the Knowledge Base

This survey grounds on the knowledge base from my previous work [23], which consists of a number of literature studies addressing performance concerns of SaSs while spanning over several other dimensions.

In particular, I have considered two surveys, one by Weyns et al. [3] and one by Becker et al. [6], which reported the state-of-art on addressing non-functional concerns by means of formal notations and MDE, respectively, until 2012. While doing this, I have also taken into account possible evolutions/extensions of the considered approaches that might have introduced new features or improved the existing ones. Additionally, I have considered a more recent systematic study by Shevtsov et al. [5], which reviewed the literature with respect to approaches exploiting CT to introduce self-adaptation and provide formal non-functional guarantees.

Among the approaches included in those three studies, five exploit the QN paradigm to address performance modeling and analysis, namely SimuLizar [18], QoSMOS [19, 27], SAFCA [20, 28], ICAC [14] and Adaptive Queuing Networks (AQNs) [9, 10] [1]. Furthermore, by additional search performed on Google Scholar [2] and Scopus [3], two more recent approaches have been identified, i.e. the ones from Incerto et al. [11] and the other one named `SMAPEA` QNs [22, 29] [4].

2.2. A Taxonomy for Classifying the Surveyed Approaches

In this section preparatory terms which can facilitate the comparison of different Performance Engineering approaches are introduced. To this aim, the different classification schemes introduced by the three systematic studies in the knowledge base, i.e. [3], [6] and [5], are taken as inspiration. As a result, the feature diagram [30] in Fig. 2 is devised, defining the following top-level categories of interest for Performance Engineering of Self-adaptive Systems (PESaSs), which are detailed in Table 1:

- *Meta-data*: This category reports the reference published papers and possible literature studies that have included the approach.
- *System architecture*: This category identifies the type of applications addressed by the approach and the exploited modeling language for representing the system.
- *Performance analysis*: This category characterizes the approaches with respect to the adopted (QN-based) performance analysis notations and methods, as well as additional artifacts and possible transformations which are needed in order to support the analysis.
- *Adaptation*: This category characterizes the adaptation mechanisms enabled by the approaches, based on typical aspects that have to be considered while introducing them, e.g. goals, knobs, inputs that can be monitored but not affected, etc.
- *Time of application*: This category is aimed at assessing if an approach can be applied at design- and/or run-time.
- *Applicability*: This category characterizes the approaches in terms of available tool-support and the provided validation.

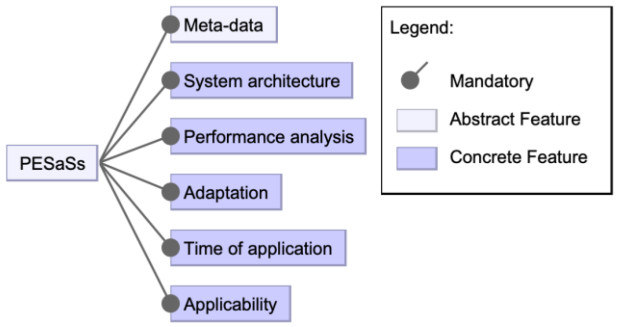

Fig. 2. Feature diagram representing the top-level categories for Performance Engineering of Self-adaptive Systems.

[1] Since the approach in [14] is unnamed, ICAC acronym is introduced from its publication venue, i.e. the International Conference on Autonomic Computing.

[2] https://scholar.google.it/

[3] https://www.scopus.com/

[4] Since the approach in [11] is unnamed, EMPC acronym is introduced from its control technique, i.e. Efficient Model Predictive Control.

Table 1. Taxonomy for Performance Engineering of Self-adaptive Systems.

Classification dimension	Description
Meta-data	
References	This reports the reference papers for the corresponding approach.
Literature studies	Systematic studies and surveys where the approach has been identified.
System architecture	
Architecture paradigm	The considered approaches can be based on *components*, *services*, *concurrent* architectures or *multi-tier* applications.
Modeling notation	This reports the modeling language adopted to represent the system architecture and to enable self-adaptation. It may coincide with performance analysis modeling notation.
Performance analysis	
Modeling notation	This reports the language adopted to represent a performance model for the self-adaptive system. As QNs represent the main foundational paradigm of our approach, we restrict to approaches exploiting QNs [8] or a particular extension of the latter, namely *Layered Queuing Networks* (LQNs) [24], as performance analysis model.
Method	The examined approaches may adopt performance models that can be solved analytically or by simulation [31].
Additional models	This reports whether pivotal representations of the system are exploited for performance analysis aims.
Transformation	Pivotal representations of the system may be derived from or used to generate – automatically, semi-automatically or manually – analysis models. To this aim, model-to-model or model-to-text transformations can be exploited. The provisioning of such transformations represents a classification criterion.
Adaptation	
Type	Conforming to Shevtsov et al. [5], four different types of adaptation can be devised: 1. *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". 2. *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". 3. *Mode adaptation* "refers to a variation in the mode of operation, which can be either *mode change* or *mode switch*. 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". 4. *Architecture reconfiguration* "refers to a run-time adaptation of the architectural structure or behavior of the application", which basically means selecting an (optimal) alternative system architecture and actually re-arranging the current one conforming to implement the former.
Controlled variables (Goal)	This category corresponds to the non-functional indices that compose the "fitness function", i.e. the goal defined by non-functional requirements. For example, response times, throughputs, utilizations, etc.
Control variables (Knobs)	This category reports "what is adapted", i.e. the predefined knobs that allow to tune the system model in order to perform adaptation. For example, concurrency level, CPU capacity allocation, service quality levels, routing probabilities, component service rates, etc.
Disturbances	This category corresponds to system parameters that may be observed but not influenced. Hence, they represent what the system has to face by self-adaptation in order to satisfy the predefined goals.
Means	Self-adaptation may be enabled by means of different techniques, e.g. Machine Learning, Control Theory, etc.
Pro-/Reactive	As from Becker et al. [6], an adaptation strategy is *reactive* if the system triggers its self-adaptation when a goal is already violated. If the system predicts that it might miss a goal sometime in the near future and hence adapts itself preventively, the adaptation strategy is *proactive*.
Time of application	
Design-/Run-time	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.
Applicability	
MDE tools	This category includes features for modeling the system and/or transform the system model into analysis models.
Analysis tools	This category represents whether the non-functional analysis is supported by existing and/or ad-hoc tools.
Optimization tools	The availability of tools for optimizing the adaptation represents a discriminant for the examined approaches.
Proof-of-concept	Approaches can be evaluated in terms of *formal correctness* (i.e. formal demonstrations are provided), *empirical correctness* (i.e. the system behaves as expected under controlled conditions) or *realism* (i.e. input/output parameters of the analysis model are similar to the ones of the actual system implementation).
Case study	The approach validity is illustrated based on a real system design or implementation in a certain (reported) domain.

3. Classification

Table 2 classifies the seven considered approaches based on the classification scheme devised in Table 1 and described in Section 2.2. In the following sub-sections, the surveyed approaches are described by looking at the columns of Table 2 (Sec. 3.1), whilst their comparison basically corresponds to the description of the rows of Table 2 (Sec. 3.2).

3.1. Description of the Surveyed Approaches

SimuLizar [18] grounds on the Palladio Component Model (PCM) [32] and extends it with a self-adaptation viewpoint and a simulation engine (namely ProtoCom [33]). A QN model is automatically derived from the PCM model by means of a model-to-model (M2M) transformation. Such QN model is simulated in order to obtain system response time. Self-adaptation viewpoints are specified in terms of Story Diagrams [34] exploiting tags (stereotypes) such as <<assign>>, <<timing>> and <<++>>, which are manually specified for elements involved in adaptation. The goal is to analyze system response time in the transient phase, after the (reactive) adaptation, under different workload intensities.

SimuLizar represents a Model-Driven performance modeling and analysis tool that can be applied at design-time, without any kind of optimization for the adaptation mechanism. Its empirical correctness has been proven with respect to a load balancing case study and a prototypal implementation has been developed, aimed at demonstrating its realism.

QoSMOS [19, 27] is a framework which enables the dynamic selection of services to face run-time changes, in the context of service-based systems. The latter are thus modeled in BPEL [35] and automatically transformed (in some unspecified way) into QN and Markov models, for performance and reliability purposes, respectively.

Proactive self-adaptation is in terms of architecture reconfiguration that, in this context, consists of dynamic service selection and resource allocation. The goal is to optimize an arbitrary QoS utility function which involves both performance and reliability requirements, e.g. system response time, system failure probability, etc., aimed at facing the operational profile [36], which specifies the workload and service failure rates. Such an optimization grounds on an algorithm which performs exhaustive search [37], whilst Bayesian estimation [38] is exploited in order to parameterize the system model with realistic values taken from the actual implementation. This latter feature allowed to prove approach realism, beside its empirical correctness.

QoSMOS is implemented a set of existing tools working in synergy, i.e. KAMI [27], PRISM [39], ProProST [40] and GPAC [41], and it has been validated with respect to a case study represented by a Tele Assistance system.

SAFCA [20, 28] enables self-adaptation of distributed and concurrent software architectures, by switching among predefined queuing patterns – namely Dynamic Thread Creation [42], HS/HA and Leader-followers [43] – at run-time, in the context of Layered QNs. The goal is to reach acceptable system response time and decrease packet loss ratio, while maximizing the utilization of software resources. To this aim, the system has to react to workload bursts, excessive queue components queue length and failure occurrences. Hence, queue lengths are monitored for performance, whilst the ratio between arrival rate and system throughput is considered for reliability; based on predefined thresholds, the adaptation is triggered by SAFCA. Although empirical correctness and realism have been proven, no tools are publicly available to the community and no case studies have been presented.

ICAC [14] generates rulesets representing adaptation policies for multi-tier architectures modeled as Layered QNs.

Self-adaptation takes the form of architectural reconfiguration taking place through knobs consisting of component replication level, CPU capacity and components allocation. The former two knobs are tuned by exploiting a gradient-based search algorithm [44], whilst component allocation is formulated as a bin-packing problem [45] which is faced by means of the *n log n* time first-fit decreasing algorithm [46]. An ad-hoc configuration optimizer is able to produce an optimal configuration for a given workload.

The configuration optimizer is used by a decision-tree learner [47] (i.e. a ML technique) at design-time, in order to obtain optimal component replication levels and CPU capacities for different workload intensities. The obtained configurations allow the decision-tree learner to generate the rulesets aimed at optimizing an arbitrary QoS utility function [5].

The approach has been validated with respect to RUBiS auction system [48], demonstrating that the response times predicted by the model correspond well with the measured response times (realism) and that the configurations carried out by the configuration optimizer are close to optimal, as well as the rulesets generated by the decision-tree learner (empirical correctness).

Concerning tool support, LQNs are solved analytically by means of the LQNS tool [49]. Moreover, the Weka tool [50], which has brought authors to adopt a decision-tree learner, might be used to investigate different ML techniques.

AQNs [9, 10] represent a particular family of QNs that allows to equip system components (i.e. service centers) with local Proportional Integral Controllers (PIDs) [51] that can adapt component's service quality level over a discrete set of predefined service demands, based on the queue length, thus resulting into an adaptation based on mode change [6]. The goal is to maintain components' queue lengths at predefined values – namely *setpoints* – representing (local) performance requirements, by reacting to workload fluctuations and unpredicted changes of the operational profile (i.e. the probability that a processed request re-enters the system).

AQNs have been implemented into the Modelica framework [53], which provides CT facilities and a simulation engine. Furthermore, a library of predefined modules has been released to ease system modeling and thus AQNs adoption by the community [10].

Finally, AQNs correctness has been proven both formally and empirically, with respect to the design of a system which provides itineraries with different level of details (service quality levels).

EMPC [11] exploits an Efficient Model Predictive Control technique [54] – namely *receding horizon* [55] – to enable proactive performance-driven self-adaptation mechanisms within QNs representing component-based systems. To this aim, formal models are exploited at run-time, such as Ordinary Differential Equations (ODEs) [56] and Mixed Integer Programming (MIP) [57], which allow to synthesize controllers implementing optimal adaptation strategies in terms of routing probabilities, components service rates and concurrency level (i.e. knobs). Model-to-text (M2T) transformation is exploited in order to obtain formal specifications from QN models.

The goal is the optimal fulfillment of performance requirements for the indices of interest (e.g. components queue lengths and/or utilization, system throughput and/or response time) through proactive prediction of workload fluctuations and service degradation.

The approach has been extensively validated with respect to a prototypal implementation of a load balancer. Correctness of formal specifications has been demonstrated, as well as the empirical correctness and the suitability of the QN model in predicting the trends of the real system (realism). Furthermore, the scalability of the approach has been evaluated and a comparison of the MIP formulation to an equivalent Markov model has been provided.

The well-known CPLEX tool can be exploited for solving MIP optimization problems, however no further tool support is available, as the approach heavily grounds on the development of scripts to execute.

`SMAPEA` QNs [22, 29] represent a novel family of QN models which allows to model and assess the performance of component-based SaSs [29]. Advanced modeling constructs

[5] Notice that the only QoS utility function considered so far is system mean response time.

[6] The concept of system *mode* is known since more than 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 [52].

such as fork/join and class-switches are exploited in order to suitably represent the managed and managing subsystems of SaSs, as well as the occurring intra- and inter-dynamics.

The SaS is assumed to be able to operate based on different (mutually exclusive) modes – e.g. *normal* and *critical* – thus devising a mode profile [52]. This enables mode-switch adaptation in the transient phase, by considering mode-profile probabilities as knobs.

A further dimension for adaptation in SMAPEA QNs has been enabled recently [22], by introducing the Controller Selection Policy (CSP) optimization problem, which concerns the routing probabilities defining how the requests are forwarded to the controllers within the managing subsystem. The goal is to find routing probabilities bringing to the optimal system's response time (in the transient phase) for each mode. To this aim, Search-Based Multi-Objective Optimization [13] is exploited, as a custom NSGA-II genetic algorithm [58].

Empirical correctness of SMAPEA QNs and the related optimization approach has been proven with respect to a realistic SaS for emergency response.

SMAPEA QNs can be developed by means of JSimGraph from the JMT tool-suite [59]; the CSP problem can be solved by exploiting a publicly available tool, namely *smapeaqn.moo*.

3.2. Comparison of the Surveyed Approaches

All the considered approaches exploit the QN paradigm for performance modeling and analysis notation, in the form of classic QNs [8] or their specializations (ad-hoc or standard, such as LQNs [24]). This characteristic represents the fundamental inclusion criteria in the knowledge base and, consequently, it defines the focus of this paper.

In most of the approaches, the architectural notation coincides with the performance notation, hence adaptation mechanisms are directly enabled within the QNs, possibly involving M2T transformation to solve an optimization problem (EMPC).

Instead, the remaining approaches – i.e. SimuLizar and QoSMOS – exploit a different modeling notation for representing SaS architecture – i.e. PCM [32] and BPEL [35], respectively. M2M transformation is exploited to (automatically or manually) obtain QNs from architecture models devising different system configurations. Hence, QNs usage is limited to performance indices estimation, while adaptation takes place at the architectural model side.

Working at this side allows to address component-based architectures (SimuLizar) as well as architectural paradigms at a higher level of abstraction, e.g. service-based (QoSMOS).

Approaches exploiting classical QNs as architectural and performance notation address component-based architectures, whilst the ones exploiting LQNs focus on concurrent (SAFCA) and multi-tier (ICAC) architectures.

A common characteristic of approaches relying on QN simulation rather than analytic resolution is that they are applied at design-time. Instead, proactive adaptation is addressed at run-time and QNs are solved analytically, as simulation might take too long in contexts where QoS requirements must be fulfilled while the SaS is running.

The architectural notation affects the adaptation mechanisms that can be enabled, especially in terms of modifiable knobs [60]. For example, SimuLizar, which makes extensive use of MDE, grounds on *stereotypes* and *tagged values* of a UML-like profiling mechanism [61] to specify adaptation conditions and the corresponding architecture model changes.

Instead, approaches directly on QNs tend to enable adaptation of the QN stations' service demands, in terms of CPU capacity allocation (ICAC), service quality levels (AQNs) or service rates (EMPC). However, other knobs at component level are devised by those approaches, in order to regulate concurrency (EMPC), replicas and their placement (ICAC), at run-time. Moreover, as QNs are stochastic models, some probabilities can represent knobs, such as routing (EMPC, SMAPEA QNs, SimuLizar) and mode-switching (SMAPEA QNs) probabilities.

Commonalities can be observed concerning the adaptation goals the system has to reach and the source of uncertainty it has to deal with.

Workload variations are considered by all the approaches as a primary source of uncertainty. In addition, SAFCA considers components queue lengths, EMPC considers hardware degradation, while AQNs and SMAPEA QNs involve aspects related to the system's operational profile – i.e. probability for a request to re-enter the system after being served (AQNs) and mode-switching probabilities (SMAPEA QNs). Furthermore, approaches addressing performance and reliability consider additional sources of uncertainty that are component failure rates (QoSMOS) and occurrences (SAFCA).

In all the approaches except AQNs, system response time (RT) represents an adaptation goal: in some cases, it is the only goal (SimuLizar, SMAPEA QNs); in other cases, it can be considered in conjunction to additional performance (ICAC, EMPC) or reliability (QoSMOS, SAFCA) indices. Among the former indices, requirements on components queue lengths are defined in several approaches, i.e. QoSMOS, EMPC and AQNs.

Goals are achieved by means of some "intelligence" that optimizes adaptation. Search-based techniques are exploited by QoSMOS (exhaustive search algorithms [37]), SMAPEA QNs (genetic algorithms [58]) and ICAC (gradient-based search [44], in conjunction with decision-tree learning [47]).

Control-based techniques seem particularly suitable to address requirements on queue lengths, as demonstrated by AQNs – through Proportional Integral Controllers (PIDs) [51] – and EMPC [54] – through receding horizon [55] in conjunction with MIP [57]. However, ad-hoc techniques can be also developed to this aim, like SAFCA-Q and SAFCA-R.

All the approaches validate empirical correctness and most of them – especially the ones applicable at run-time – prove realism with respect to an actual SaS implementation (e.g. QoSMOS validates Bayesian estimation [38] for model parameterization) – not provided by AQNs and SMAPEA QNs.

Furthermore, approaches relying on Control Theory techniques – i.e. ANQs and EMPC – prove the formal correctness of the latter. Evaluation is provided with respect to a case study by all the approaches except SAFCA, which is limited to a proof-of-concepts of the proposed queuing patterns. For this reason, SAFCA does not provide any tool support, which is instead provided by other approaches in different forms and at different extents. In particular, QoSMOS, ICAC SimuLizar and SMAPEA QNs are supported by publicly available tools: QoSMOS is a tool chain involving KAMI [27], PRISM [39], ProProST [40] and GPAC [41] for the MAPE loop at run-time; ICAC exploits the LQNS solver [49] for performance analysis and the Weka tool [50] for optimization purposes; SimuLizar uses PCM models [32] and story diagrams [34] for modeling and the ProtoCom engine [33] for analysis; SMAPEA QNs are entirely supported by JSimGraph from JMT tool-suite [59] for performance modeling and analysis and provides a multi-objective optimization tool for the CSP problem.

No particular tools are explicitly devised to apply EMPC, as it envisions the development of Python scripts to be executed at run-time. However, the CPLEX tool can be exploited to solve MIP formulation. Finally, differently from other approaches, AQNs have been developed within the Modelica framework from scratch, resulting into a library of modeling components that can be used to build system representations.

Table 2. Classification of the surveyed approaches.

Category	**SimuLizar**	**QoSMOS**	**SAFCA**	**ICAC**	**AQNs**	**EMPC**	**SMAPEA QNs**
Meta-data							
References	[18]	[19], [27]	[20], [28]	[14]	[9], [10]	[11]	[22], [29]
Literature studies	[6]	[6], [3]	[6]	[3]	[5]	-	-
System Architecture							
Architecture paradigm	Component-based	Service-based	Concurrent	Multi-tier	Component-based	Component-based	Component-based
Modeling notation	PCM	BPEL	LQN	LQN	QN	QN	QN
Performance analysis							
Modeling notation	QN	QN	LQN	LQN	QN	QN	QN
Method	Simulative	Analytical	Analytical	Analytical	Simulative	Analytical	Simulative
Additional models	No	Markov Models	No	No	No	ODE, MIP	No
Transformation	M2M	Unspecified	No	No	No	M2T	No
Adaptation							
Type	Arch. reconfig.	Arch. reconfig.	Arch. reconfig.	Arch. reconfig.	Mode-change	Comp./Par. change	Mode-switch,
Goals	System RT	QoS utility function (Fail. Prob., Comp. queue len., System RT)	System RT, Packet loss ratio, Sw resource utils,	QoS utility function (System RT)	Component queue lengths	Component queue lengths, Throughput, Utilizations, System RT)	System modes' RTs
Knobs	Elements tagged with <<assign>>, <<timing>> or <<++>> stereotypes	Service selection, CPU capacity allocation	Queue patterns (HS/HA, Leader-followers, Dynamic Thread Creation)	Component replication level, CPU capacity allocation, Components placement	Comp. service quality levels	Routing probabilities, Comp. service rates, Concurrency level	Mode-switching probabilities, Controller Selection Policies
Sources of uncertainty	Workload variations	Operational model (Service failure rates, Workload variations)	Component queue lengths, Workload variations, Failure occurrence	Workload variations	Workload variations, Jobs exit probability	Workload variations, Hardware degradation	Workload variations, Mode-switching probabilities
Means	Story diagrams	Bayesian estimation (params. estimation), Exhaustive search algorithms	Thresholds on queue lengths (SAFCA-Q), Arrival rate Vs System throughput (SAFCA-R)	Decision trees, Gradient-based search	Prop. Integral Controllers	Receding horizon, Mixed Integer Programming	Mode profiling, Genetic algorithms
Pro-/Reactive	reactive	proactive	reactive	reactive	reactive	proactive	both
Time of application							
Design-/Run-time	design-time	run-time	run-time	design-time	design-time	run-time	both
Applicability							
MDE tools	PCM	KAMI, PRISM, GPAC	No	No	Modelica	No	JSimGraph
Analysis tools	ProtoCom	PRISM, ProProST	No	LQNS	Modelica	No	JMT
Optimization tools	No	GPAC	No	Weka	Modelica	CPLEX	smapeaqn.moo
Proof-of-concept	Empirical correctness, Realism (prototypal)	Empirical correctness, Realism	Empirical correctness, Realism	Empirical correctness, Realism	Formal correctness, Empirical correctness	Formal correctness, Empirical correctness, Realism (prototypal)	Empirical correctness
Case study	Load balancing	Tele Assistance	No	RUBiS	Itineraries provision	Load balancing	Emergency handling

4. Lessons Learned

The following key-points summarize the main findings resulting from this literature study.

- System response time is the most addressed performance metric.
- Typically, non-functional goals must be fulfilled while facing workload variations.
- Reactive adaptation is usually addressed at design-time by simulation, whilst proactiveness is typically addressed at run-time by analytic resolution. In fact, managing simulation overhead while addressing run-time adaptation might be costly.
- MDE can provide useful support for performance modeling and analysis of SaSs, however it seems to be particularly suitable at design-time only.
- Control Theory can be successfully applied to provide formal guarantees by introducing global or local controllers.
- Optimization techniques such as Machine Learning, Search-Space Exploration and Mixed Integer Programming, can be exploited in order to optimize a fitness function involving performance indices.
- Empirical validation with respect to a case study is the basic form of evaluation which is typically provided. Besides, exploiting control-based techniques implies a need for formal validation, which represents an added value.
- Actual system implementations are likely used in order to parameterize analysis models in a realistic way and/or to compare analysis results to measurements from the running system.
- The availability of modeling and analysis tools, as well as benchmark systems implementations is crucial for the adoption of any approach for self-adaptation. Relying on existing widespread tools and system implementations represents a valuable choice, as ad-hoc development can be very costly.

5. Conclusion

In this paper, I have extended previous work [ANT2020] which surveyed the literature with respect to approaches enabling performance-driven self-adaptation supported by the Queuing Network paradigm. The classification scheme that have been previously introduced has been revised in order to carry out a taxonomy which allowed to detail the considered approaches spanning among different dimensions, with particular emphasis on the ways adaptation mechanisms that have been introduced and their non-functional goals.
Internal characteristics of those approaches have been described, as well as their commonalities and differences, aimed at clarifying the state-of-art in addressing self-adaptation by exploiting QNs. Hence, this work can be used to get a detailed view of the current state-of-art in this context.

Acknowledgments

This work is supported by the Italian Ministry of Education, University and Research -- MIUR, L. 297, art. 10.
The author would like to thank Dott. Davide Di Ruscio for his precious suggestions concerning paper presentation.

References

[1] Perez-Palacin D, Mirandola R. Uncertainties in the modeling of self-adaptive systems: a taxonomy and an example of availability evaluation. Proceedings of the 5th ACM/SPEC International Conference on Performance Engineering. Dublin, IE, 2014. https://doi.org/10.1145/2568088.2568095

[2] Camara J, Garlan D, Kang WG, Peng W, Schmerl B.R. Uncertainty in Self-Adaptive Systems Categories, Management, and Perspectives. Carnegie Mellon University Tech Report CMU-ISR-17-110, July 2017.

[3] Weyns D, Iftikhar MU, de la Iglesia DG, Ahmad T. 2012. A survey of formal methods in self-adaptive systems. Proceedings of the 5th International C* Conference on Computer Science and Software Engineering. Montréal, CA, 2012. https://doi.org/10.1145/2347583.2347592

[4] Kephart JO, Chess DM. The vision of autonomic computing. IEEE Computer 2003;36:41–50. https://doi.org/10.1109/MC.2003.1160055

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

[6] Becker M, Luckey M, Becker S. Model-driven performance engineering of self-adaptive systems: A survey. Proceedings of the 8th International ACM SIGSOFT conference on Quality of Software Architectures. Bertinoro, IT, 2012. https://doi.org/10.1145/2304696.2304716

[7] Hellerstein J, Diao Y, Parekh S, Tilbury D. Feedback Control of Computing Systems. Wiley, 2004. https://doi.org/10.1002/047166880X

[8] Lazowska ED, Zahorjan J, Graham GS, Sevcik KC. Quantitative system performance - computer system analysis using queueing network models. Prentice Hall, 1984.

[9] Arcelli D, Cortellessa V, Filieri A, Leva A. Control theory for model-based performance-driven software adaptation, Proceedings of the 11th International ACM SIGSOFT Conference on Quality of Software Architectures. Montréal, CA, 2015. https://doi.org/10.1145/2737182.2737187

[10] Arcelli D, Cortellessa V, Leva A. A library of modeling components for adaptive queuing networks. Proceedings of the 13th European Workshop on Performance Engineering. Chios, GR, 2016. https://doi.org/10.1007/978-3-319-46433-6_14

[11] Incerto E, Tribastone M, Trubiani C. Software performance self-adaptation through efficient model predictive control. 32nd IEEE/ACM International Conference on Automated Software Engineering. Urbana, IL, 2017. https://doi.org/10.1109/ASE.2017.8115660

[12] James G, Witten D, Hastie T, Tibshirani R. An Introduction to Statistical Learning: With Applications in R. Springer, 2014. https://doi.org/10.1007/978-1-4614-7138-7

[13] Harman M, Afshin Mansouri S, Zhang Y. Search-based software engineering: Trends, techniques and applications.

ACM Computing Surveys 2012;45(1):11:1–11:61. https://doi.org/10.1145/2379776.2379787

[14] Jung G, Joshi KR, Hiltunen MA, Schlichting RD, Pu C. Generating adaptation policies for multi-tier applications in consolidated server environments. Proceedings of the 5th International Conference on Autonomic Computing. Chicago, IL, 2008. https://doi.org/10.1109/ICAC.2008.21

[15] Elkhodary A, Esfahani N, Malek S. Fusion: A framework for engineering self-tuning self-adaptive software systems. Proceedings of the 18th ACM SIGSOFT Symposium on the Foundations of Software Engineering. Santa Fe New Mexico, USA, 2010. https://doi.org/10.1145/1882291.1882296

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

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

[18] Becker M, Becker S, Meyer J. Simulizar: Design-time modeling and performance analysis of self-adaptive systems. In: Proceedings of Software Engineering 2013. Aachen, DE, 2013.

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

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

[21] Kounev S, Brosig F, Huber N, Reussner RH. Towards self-aware performance and resource management in modern service-oriented systems. Proceedings of the 7th IEEE International Conference on Services Computing. Miami, FL, 2010. https://doi.org/10.1109/SCC.2010.94

[22] Arcelli D. A Multi-Objective Performance Optimization Approach for Self-Adaptive Architectures. Proceedings of the 14th European Conference on Software Architecture. L'Aquila, IT, 2020. https://doi.org/10.1007/978-3-030-58923-3_9

[23] Arcelli D. Exploiting Queuing Networks to Model and Assess the Performance of Self-Adaptive Software Systems: A Survey. Proceedings of the 11th International Conference on Ambient Systems, Networks and Technologies. Warsaw, PL, 2020. https://doi.org/10.1016/j.procs.2020.03.108

[24] Franks G, Majumdar S, Neilson J, Petriu D, Rolia J, Woodside M. Performance analysis of distributed server systems. Proceedings of the 6th International Conference on Software Quality, 1996.

[25] Puterman ML. Markov Decision Processes. Wiley, 1994. https://doi.org/10.1002/9780470316887

[26] Reisig W. Petri nets: an introduction. Springer, 1985. https://doi.org/10.1007/978-3-642-69968-9

[27] Epifani I, Ghezzi C, Mirandola R, Tamburrelli G. Model evolution by run-time parameter adaptation. Proceedings of the 31st International Conference on Software Engineering. Vancouver, CA, 2009. https://doi.org/10.1109/ICSE.2009.5070513

[28] Zhang X, Lung C, Franks G. Towards architecture-based autonomic software performance engineering. Proceedings of the 4th French-speaking Conference on Software Architectures. Pau, FR, 2010.

[29] Arcelli D. Towards a Generalized Queuing Network Model for Self-adaptive Software Systems. Proceedings of the 8th International Conference on Model-Driven Engineering and Software Development. Valletta, Malta, 2020. https://doi.org/10.5220/0009180304570464

[30] Batory D. Feature Models, Grammars, and Propositional Formulas. Proceedings of the 9th International Conference on Software Product Lines. Rennes, FR, 2005. https://doi.org/10.1007/11554844_3

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

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

[33] Becker S, Dencker T, Happe J. Model-driven generation of performance prototypes. Proceedings of the SPEC International Performance Evaluation Workshop. Darmstadt, DE, 2008.

[34] Von Detten M, Heinzemann C, Platenius MC, Rieke J, Travkin D, Hildebrandt S. Story Diagrams Syntax and Semantics. Heinz Nixdorf Institute Tech Report tr-ri-12-324, July 2012.

[35] Juric MB, Mathew B, Sarang P. Business Process Execution Language for Web Services. Packt Publishing, 2004.

[36] Musa JD. Operational profiles in software-reliability engineering. IEEE Software 1993;10:14–32. https://doi.org/10.1109/52.199724

[37] Calinescu R, Kwiatkowska MZ. Using quantitative analysis to implement autonomic IT systems. Proceedings of the IEEE 31st International Conference on Software Engineering. Vancouver, CA, 2009. https://doi.org/10.1109/ICSE.2009.5070512

[38] Berger JO. Statistical Decision Theory and Bayesian Analysis, 2nd edition. Springer, 1985. https://doi.org/10.1007/978-1-4757-4286-2

[39] Kwiatkowska MZ, Norman G, Parker D. Probabilistic symbolic model checking with PRISM: A hybrid approach, Int. J. on Software Tools for Technology Transfer 2004;6(2):128–142. https://doi.org/10.1007/s10009-004-0140-2

[40] Grunske L. Specification patterns for probabilistic quality properties. 30th International Conference on Software

Engineering. Leipzig, DE, 2008. https://doi.org/10.1145/1368088.1368094

[41] Calinescu R. General-purpose autonomic computing. In: Zhang Y, Yang L, Denko M (Eds), Autonomic Computing and Networking, Springer, 2009, pp. 3–30. https://doi.org/10.1007/978-0-387-89828-5_1

[42] Welsh M, Gribble S, Brewer E, Culler D. A Design Framework for Highly Concurrent Systems. UC Berkley Tech Report UCB/CSD-00-1108, January 2000.

[43] Schmidt D, Stal M, Rohnert H, Buschmann F. Pattern-Oriented Software Architecture, Patterns for Concurrent and Networked Objects. Wiley, 2000.

[44] Curry HB. The Method of Steepest Descent for Non-linear Minimization Problems. Quarterly of Applied Mathematics 1944;2(3):258–261. https://doi.org/10.1090/qam/10667

[45] Chekuri C, Khanna S. On multidimensional packing problems. SIAM J. on Computing 2004;33(4):837–851. https://doi.org/10.1137/S0097539799356265

[46] Coffman Jr EG, Galambos G, Martello S, Vigo D. Bin Packing Approximation Algorithms: Combinatorial Analysis. In: Du D-Z, Pardalos PM (Eds), Handbook of Combinatorial Optimization. Kulwer, 1998, pp. 151–207. https://doi.org/10.1007/978-1-4757-3023-4_3

[47] Shalev-Shwartz S, Ben-David S. Decision Trees In: Understanding Machine Learning: From Theory to Algorithms. Cambridge University Press, 2014, pp. 212–218. https://doi.org/10.1017/CBO9781107298019.019

[48] Cecchet E, Chanda A, Elnikety S, Marguerite J, Zwaenepoel W. Performance comparison of middleware architectures for generating dynamic web content. Proceedings of the 4th ACM/IFIP/USENIX International Middleware Conference. Rio de Janeiro, BR, 2003. https://doi.org/10.1007/3-540-44892-6_13

[49] Franks G, Maly P, Woodside M, Petriu DC, Hubbard A, Mroz M. Layered Queueing Network Solver and Simulator User Manual. Real-time and Distributed Systems Lab, Carleton University, January 2013.

[50] Frank E, Hall MA, Holmes G, Kirkby R, Pfahringer B, Witten IH, Trigg L. Weka-A Machine Learning Workbench for Data Mining. In: Maimon O, Rokach L (Eds), Data Mining and Knowledge Discovery Handbook 2nd edition. Springer, 2010, pp. 1269–1277. https://doi.org/10.1007/978-0-387-09823-4_66

[51] Åström KJ, Hägglund T. Advanced Pid Control. ISA-The Instrumentation, Systems, and Automation Society, 2006.

[52] Morin B, Barais O, Nain G, Jézéquel J. Taming dynamically adaptive systems using models and aspects. Proceedings of the 31st International Conference on Software Engineering. Vancouver, CA, 2009. https://doi.org/10.1109/ICSE.2009.5070514

[53] Fritzson P, Engelson V. Modelica — A unified object-oriented language for system modeling and simulation. Proceedings of the 12th European Conference on Object-Oriented Programming. Brussels, BE, 1998. https://doi.org/10.1007/BFb0054087

[54] García CE, Prett DM, Morari M. Model predictive control: Theory and practice—a survey. Automatica 1989;25(3):335–348. https://doi.org/10.1016/0005-1098(89)90002-2

[55] Abdelwahed S, Bai J, Su R, Kandasamy N. On the application of predictive control techniques for adaptive performance management of computing systems. IEEE Trans. on Network and Serv. Manag. 2009;6(4):212–225. https://doi.org/10.1109/TNSM.2009.04.090402

[56] Kurtz TG. Solutions of ordinary differential equations as limits of pure Markov processes. Journal of Applied Probability 1970;7(1):49–58. https://doi.org/10.2307/3212147

[57] Achterberg T, Wunderling R. Mixed integer programming: Analyzing 12 years of progress. In: Junger M, Reinelt G (Eds), Facets of Combinatorial Optimization, Springer, 2013, pp. 449–481. https://doi.org/10.1007/978-3-642-38189-8_18

[58] Deb K, Pratap A, Agarwal S, Meyarivan T. A fast and elitist multiobjective genetic algorithm: Nsga-ii. IEEE Trans. on Evolutionary Computation 2002;6(2):182–197. https://doi.org/10.1109/4235.996017

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

[60] Arcelli D, Cortellessa V. Software model refactoring based on performance analysis: better working on software or performance side?. Proceedings of the 10th International Workshop on Formal Engineering Approaches to Software Components and Architecture. Rome, IT, 2013. https://doi.org/10.4204/EPTCS.108.3

[61] Alhir SS. Extension Mechanisms. In: Guide to Applying the UML. Springer Professional Computing, 2002, pp. 343–363. https://doi.org/10.1007/0-387-21513-1_10