Towards a Generalized Queuing Network Model for Self-adaptive Software Systems

Davide Arcelli[1,2][a]
[1]*Università degli Studi dell'Aquila, via Vetoio 1, L'Aquila, Italy*
[2]*Department of Information Engineering, Computer Science and Mathematics, L'Aquila, Italy*

Keywords: Self-adaptive Software Systems, Software Architecture, Architectural Patterns, Performance, Queuing Networks.

Abstract: A Self-adaptive Software Systems (SASSs) is composed by a managing and a managed subsystem. The former comprises system's adaptation logic and controls the latter, which provides system's functionalities by perceiving and affecting the environment through its sensors and actuators, respectively. Such control often conforms to a MAPE-K feedback loop, i.e. a Knowledge-based architecture model that divides the adaptation process into four activities, namely Monitor, Analyze, Plan and Execute. Performance modeling notations, analysis methods and tools, have been coupled to other kinds of techniques (e.g. control theory, machine learning) for modeling and assessing the performance of managing subsystems, possibly aimed at supporting the identification of more convenient architectural alternatives. The contribution of this paper is a generalized Queuing Network model for SASSs, where the managed subsystem is explicitly modelled, thus widening performance modeling and analysis scope to the whole system. Job classes flowing through the QN represent activities of a global feedback control loop, which is based on the system's mode profile and implemented by class-switches operating in conformance to proper predefined class-switching and routing probabilities. Results obtained by means of a proof-of-concept addressing a realistic case study show that the generalized QN model can usefully support performance-driven architectural decision-making.

SCIENCE AND TECHNOLOGY PUBLICATIONS

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 (Perez-Palacin and Mirandola, 2014; Cámara et al., 2017). To this aim, much work has been done, mainly by introducing autonomic managers – namely *managing subsystems* – comprising 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 (Weyns et al., 2012). Such coupling often results into *MAPE-K feedback loop(s)* (Kephart and Chess, 2003), i.e. "an architecture model that divides the process of adaptation into four phases: Monitor (**`M`**), Analyze (**`A`**), Plan (**`P`**), and Execute (**`E`**). Data that is collected and used during adaptation is stored in the so-called Knowledge base (**`K`**)" (Becker et al., 2012).

In such a context, performance is a top concern, as from (Weyns et al., 2012; Shevtsov et al., 2018; Becker et al., 2012). 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 assessment of managing subsystems. For example, Control Theory allowed to introduce (global or local) `MAPE` feedback controllers able to provide formal guarantees within performance models such as Queuing Networks (QNs) (Arcelli et al., 2015; Arcelli et al., 2016; Incerto et al., 2017); Machine Learning and Model-Driven Engineering (MDE) have been used in heuristic or semi-formal approaches to (semi-)automatically reasoning and deciding about (read analyzing and planning) adaptation, driven by performance requirements (Jung et al., 2008; Elkhodary et al., 2010; Barati et al., 2019; Grassi et al., 2009; Becker et al., 2013; Calinescu et al., 2011; Lung et al., 2016; Kounev et al., 2010). While the math-

[a] https://orcid.org/0000-0002-8919-9831

Arcelli, D.
Towards a Generalized Queuing Network Model for Self-adaptive Software Systems.
DOI: 10.5220/0009180304570464
In *Proceedings of the 8th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2020)*, pages 457-464
ISBN: 978-989-758-400-8; ISSN: 2184-4348
Copyright © 2022 by SCITEPRESS – Science and Technology Publications, Lda. All rights reserved

ematical 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 (Weyns et al., 2012).

This paper presents an approach to enhance the casting of a Self-Adaptive Software System (SASS) as a QN (Lazowska et al., 1984), and to thus be able to leverage queuing networks analysis tools and methods in analyzing the performance of the system, in ways that were not previously possible.

In fact, differently from existing approaches, whose main goal is to introduce and validate the performance of autonomic managing subsystem(s) enabling self-adaptation mechanisms, we introduce in this paper a generalized performance model – in the form of QN – that widens the focus of performance analysis from the self-adaptation mechanisms (in charge of the managing subsystem only) to the whole SASS, thus involving sensing and actuating components belonging to the managed subsystem.

Hence, both the subsystems are abstracted by our QN models. In particular, system's components are represented by *stations*, which are visited by *job classes* corresponding to the different tasks performed by the managing and managed subsystems, i.e. `MAPE` and sensing/actuating, respectively. A `MAPE` feedback loop is implemented by exploiting advanced QN constructs supported by JMT (Bertoli et al., 2018) (i.e. a standard de-facto), namely class switches, which enable dynamic job transformation conforming to the system's mode profile (Musa, 1993).

The paper is structured as follows: Section 2 reviews existing approaches exploiting the QNs as performance notation for SASSs. Section 3 illustrates the generalized QN model and the underlying reference self-adaptation model. Section 4 provides a proof-of-concept showing our approach can usefully support performance-driven architectural decisions. Section 5 concludes the paper and points out our future intents.

2 RELATED WORK

We consider related work approaches relying on the QN paradigm for performance assessment of SASSs. In this context, (Weyns et al., 2012) and (Becker et al., 2012) surveyed the state-of-art on addressing non-functional concerns through formal notations and MDE, respectively, until 2012. A more recent study by (Shevtsov et al., 2018) reviewed approaches exploiting Control Theory, able to provide non-functional formal guarantees.

Among the approaches included in those studies, five exploit the QN paradigm to address performance modeling and analysis, namely SimuLizar (Becker et al., 2013), QoSMOS (Epifani et al., 2009; Calinescu et al., 2011), SAFCA (Zhang et al., 2010; Lung et al., 2016), ICAC (Jung et al., 2008)[1] and Adaptive Queuing Networks (AQNs) (Arcelli et al., 2015; Arcelli et al., 2016). By additional search, we identified a recent work by (Incerto et al., 2017) exploiting Efficient Model Predictive Control (EMPC[2]) to enable performance-driven self-adaptation within QNs.

SimuLizar, QoSMOS, SAFCA and ICAC introduce self-adaptation by enabling architecture reconfiguration. SimuLizar and QoSMOS enable reconfiguration onto architecture models, which implies a need for model transformation towards the QN notation; in SAFCA and ICAC, instead, reconfiguration takes place directly onto QNs, without involving any additional architecture model[3]. Both SAFCA and ICAC rely on Layered Queuing Networks (LQNs) – i.e. a well-known extension of QNs – as performance notation, which likely resulted suitable for the architectural paradigms addressed by those approaches, i.e. concurrent and multi-tier systems, respectively.

Differently from the four aforementioned approaches, EMPC and AQNs – that rely on control theory – introduce self-adaptation by automated tuning of predefined knobs (i.e. routing probabilities, concurrency level and components' service rates) in a proactive and reactive manner, respectively, aimed at satisfying global (EMPC) or local (AQNs) performance requirements. It is worth to notice that, in AQNs, the system can adapt 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-based adaptation type (Shevtsov et al., 2018). Such kind of adaptation is the one supported by our approach and it can be suitably modeled by exploiting the *class-switch* QN construct, which is able to transform jobs modulo available job classes, conforming to predefined probabilities. Such probabilities are 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* (Musa, 1993). However, since they strictly depend on the specific application domain,

[1]This approach has not been named, so we introduced the acronym ICAC from its publication venue.

[2]This acronym will be be used throughout the paper to refer (Incerto et al., 2017).

[3]Readers interested in a comparison between working at software and performance model sides may refer to (Arcelli and Cortellessa, 2013).

their setting is out from the scope of this paper, that is introducing a generalized QN model supporting performance modeling and analysis of SASSs.

Performance analysis methods of related work equally distributes between analytical and simulative. Our approach adopts the latter, due to advanced constructs (e.g. class-switches) within the QN models, which inhibit the former methods. This might limit the adoption of the approach at run-time, i.e. when several performance models are generated from an existing SASSs and analyzed to explore better performing architectural alternatives, because simulation might take long with respect to dynamics occurring at run-time, especially in the context of safety-critical systems with real-time performance requirements.

3 APPROACH DESCRIPTION

As mentioned before, the goal of this paper is to introduce a generalized QN model supporting performance modeling and analysis of SASSs, in terms of both managing and managed subsystems. To this aim, prior to the definition of the generalized QN model, we define the underlying reference model for self-adaptation, which comes from a reworking on the autonomic control loop model proposed by (Weyns et al., 2013).

3.1 Self-adaptation Model

Figure 1 illustrates the reference self-adaptation model our approach relies on. The *managed subsystem* represents an interface through which the system perceives the *environment* – by means of sensors within a *Perception Layer* (PL) – and applies onto the latter – by means of actuators within an *Application Layer* (AL); The *managing subsystem* interprets such perceptions and may consequently react with adaptation of system's behavior – by means of controllers within an *Intermediate Layer* (IL), aimed at achieving system's goals. The two subsystems are usually connected through network(s) (see the dashed line).

Conforming to (Weyns et al., 2013), adaptation logic involves MAPE control loops (Kephart and Chess, 2003) and its four sequential activities. As from Table 1, our self-adaptation model devises two additional activities, namely (0) *Sense* and (5) *Actuate*, both located within the managed subsystem. *Sense* is associated to the term "measure" to distinguish raw data retrieval from their subsequent aggregation (read "collect") performed during *Monitor*. Moreover, we associate the term "contrive" to *Plan*, whilst "prepare" and "act" shift to *Execute* and *Actuate*, respectively. We

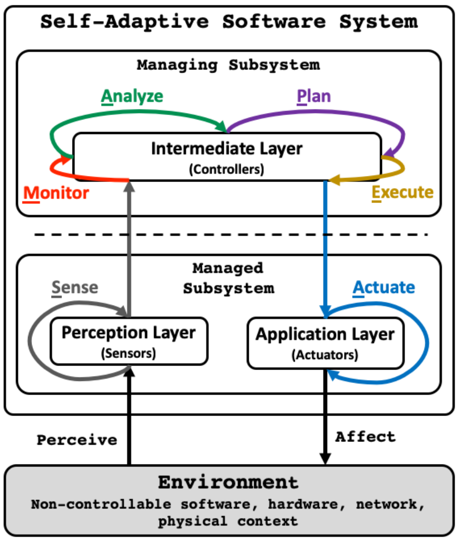


Figure 1: Reference model for self-adaptation.

think that *Plan* activity is better associated to conceiving (read "contrive") a solution rather than its preparation. The latter, in turn, remains in charge of the managing subsystem, but is associated to *Execute*.

Table 1: MAPE Vs SMAPEA control loops.

Order	Activities	
	(Weyns et al., 2013)	Our approach
0	-	Sense (measure)
1	Monitor (collect)	Monitor (collect)
2	Analyze (determine)	Analyze (determine)
3	Plan (prepare)	Plan (contrive)
4	Execute (act)	Execute (prepare)
5	-	Actuate (act)

Devising the two activities within the managed subsystem allows to widen the modeling scope to the boundaries of the system, i.e. sensing and actuating components – where interactions with the environment take place. Furthermore, (Weyns et al., 2013) directly map MAPE activities to ad-hoc components, each dedicated to perform the corresponding activity, whilst SMAPEA activities are properly performed spanning among sensors (responsible for *Sense*), controllers (responsible for *Monitor*, *Analyze*, *Plan* and *Execute*) and actuators (responsible for *Actuate*). Such distinction brings to a natural mapping of: (i) system's components onto QN *stations* and (ii) activities onto QN *job classes*, resulting into the generalized model described in the next section.

The concept of *mode* is known by more than a decade in the context of dynamic adaptive systems and it has been used to devise different configurations among which a system may transit for self-adaptation (Morin et al., 2009), based on predefined probabilities representing a mode profile (Musa, 1993). As an example, Figure 2 shows a mode profile of a SASS for emergency response.

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

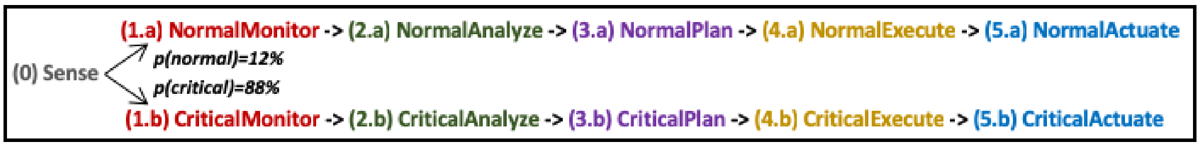

(0) Sense
p(normal)=12%
p(critical)=88%
(1.a) NormalMonitor -> (2.a) NormalAnalyze -> (3.a) NormalPlan -> (4.a) NormalExecute -> (5.a) NormalActuate
(1.b) CriticalMonitor -> (2.b) CriticalAnalyze -> (3.b) CriticalPlan -> (4.b) CriticalExecute -> (5.b) CriticalActuate

Figure 2: Example of self-adaptation within SMAPEA Queuing Networks.

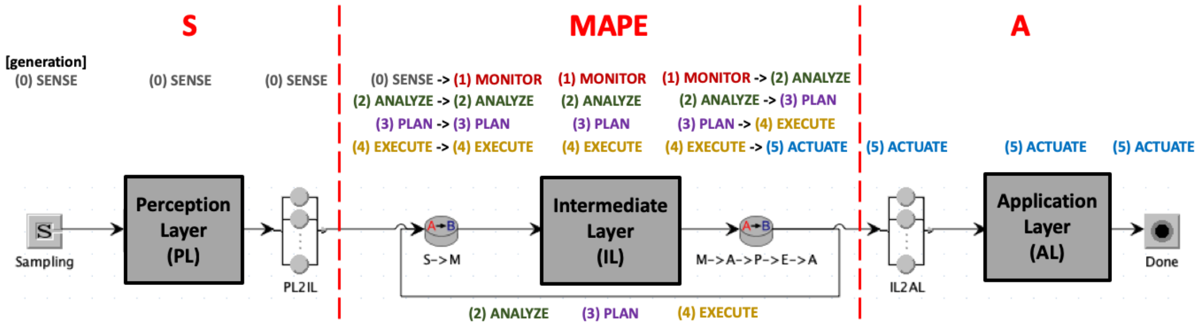


Figure 3: Generalized SMAPEA Queuing Network Model.

The system is allowed to *switch* between *normal* and *critical* modes with 12% and 88% probability, respectively. Such probabilities allow to switch from *Sense* to mode-based *Monitor* jobs, corresponding to the adoption of a particular mode. *Monitor*, *Analyze*, *Plan*, *Execute* and *Actuate* classes shall be thus "instantiated" modulo those two modes, resulting into 10 `MAPEA` instances, i.e. $\{Normal, Critical\} \times \{Monitor, Analyze, Plan, Execute, Actuate\}$.

Actuate instantiation depends on the specific type of actuations across the different system modes. For example, supposing the SASS for emergency response above has two actuating components, namely *Dashboard* and *EvacuationSigns*, four *Actuate* instances might be devised, i.e. $\{Normal, Critical\} \times \{DashboardActuate, EvacuationSignsActuate\}$. Similarly, *Sense* instantiation depends on the specific type of sensing activities. For example, assuming two sensing components, namely *Camera* and *Temperature*, two *Sense* instances might be devised, i.e. *CameraSense* and *TemperatureSense*.

3.2 Generalized Queuing Network Model for SASSs

Figure 3 depicts the generalized QN conforming to the reference self-adaptation model of Section 3.1, in terms of (i) stations and their connections and (ii) job classes visiting such stations.

The QN is partitioned in three parts, namely **S**, **MAPE** and **A**, each containing a conceptual macro-component corresponding to PL, IL and AL, respectively. The actual system's components belonging to macro-components, i.e. sensors, controllers and actuators, are mapped onto (properly connected and parameterized) *stations*. Hence, `SMAPEA` activities naturally map onto *job classes* visiting QN stations.

PL and AL, that are the parts of the managed subsystem interacting with the environment, are connected to *Sampling* and *Done*, i.e. a source and a sink node that participate to `Perceive` and `Affect` interactions, respectively (Figure 1). In particular, at the **S**-side, *Sampling* emulates environmental stimuli by generating *SENSE* jobs, based on a certain probability distribution to be specified[4]. For example, a single workload source is illustrated in Figure 3, i.e. sensors have the same sampling rate, although introducing more workload sources to differentiate sampling rates is possible. At the **A**-side, instead, *Done* represents the fact that actuation has been performed and the environment has been (possibly) affected.

Both the PL and the AL are connected to the IL – which contains controllers – through *PL2IL* and *IL2AL* delay stations, respectively. The latter represent the network(s) between the managed and the managing subsystems.

In order to implement the `SMAPEA` control loop, two *class-switch*es are introduced, namely $S{\rightarrow}M$ and $M{\rightarrow}A{\rightarrow}P{\rightarrow}E{\rightarrow}A$, placed before and after the IL, respectively. $S{\rightarrow}M$ transforms *SENSE* jobs into *MONITOR*, whilst $M{\rightarrow}A{\rightarrow}P{\rightarrow}E{\rightarrow}A$ transforms each `MAPEA` job into a job of the subsequent type.

Transformation from *SENSE* to *MONITOR* grounds on mode-switching probabilities mentioned

[4]By uppercase words we denote all *Sense*, *Monitor*, *Analyze*, *Plan*, *Execute* and *Actuate* job classes. E.g., with respect to Figure 2, $MONITOR = \{NormalMonitor, CriticalMonitor\}$.

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

in Section 3.1 and exemplified in Figure 2. Mode-specific *MONITOR* jobs are forwarded to the controllers, spending their demands before visiting $M \rightarrow A \rightarrow P \rightarrow E \rightarrow A$. The latter transforms them into *ANALYZE* jobs, which are routed back to $S \rightarrow M$. The loop is further iterated by producing the subsequent mode-specific *PLAN*, *EXECUTE* and *ACTUATE* jobs. Finally, *ACTUATE* jobs are forwarded to the **A**-side for actuation.

When a MAPE job is forwarded back to the IL, it must be decided which controller shall serve the job. This choice is part of the so-called *Controller Selection Policy* (CSP), demanded to $S \rightarrow M$. Hence, specifying a CSP for a SMAPEA QN corresponds to the specification of $S \rightarrow M$ routing strategies for each job class. Several routing strategies are available and some of them allow to introduce run-time self-adaptation. However, the latter is out of the scope of this paper, which covers probability-based routing strategies only, leaving other strategies as future work without jeopardizing paper contribution. In fact, a probability-based strategy for $S \rightarrow M$ could be exploited to implement several architectural patterns, as shown later in the paper (Section 4).

4 EVALUATION

This section provides a proof-of-concepts for the generalized QN model, by exploiting a revised version of the case study addressed in our previous work (Arbib et al., 2019). The considered SASS grounds on a reactive IoT architecture for emergency response designed for a real exhibition venue in Alan Turing Building – Department of Information Engineering, Computer Science and Mathematics – L'Aquila, Italy.

CCTV cameras detect people position and movement, used in conjunction with data from temperature and CO_2 sensors to feed an algorithm run by control components. The latter decide on the actuation set based on the situation: (i) In *normal* mode, the system shows a 2D-representation of the monitored space on a dashboard, locating crowds and tracing their movements, while periodically estimating the minimum evacuation time required under current conditions; In *critical* mode – e.g. an emergency happens needing people evacuation – additional information is shown on the dashboard, alarm actuators are activated and evacuation signs indicate the best evacuation routes.

Figure 4 shows a SMAPEA QN representing an architecture model for the case study. Note that, **S**, **MAPE** and **A** partitions, are actually delimited by forks/joins, needed to create sensor/actuator-specific jobs and properly route them to the corresponding sensing/actuating component. Beside these PL/AL components, we introduce two local controllers – namely *CentralController* and *PeerController* – and a remote controller deployed at the cloud – namely *CloudController*, placed between *Uplink* and *Downlink* delay stations (i.e. the cloud access network).

Concerning service demands distributions, we reuse most of the ones exploited in our previous work (Arbib et al., 2019), since they resulted from a deep investigation conducted by means of other simulation tools – i.e. CAPS (Muccini and Sharaf, 2017) and the well-known CPLEX[5] – which carried out transmission and propagation delays for sensing, actuating and networking components, as well as controllers' processing times for performing the different MAPE activities[6]. Notice that, the combination between high demands for planning and *IL2AL* during emergencies makes the critical mode crucial as it determines a physiological lower bound to system performance. For this reason, we assume a mode adaptation probability of 12% for *Normal* and 88% for *Critical*), which allow to stress the system during emergencies.

The (shared) sampling rate is modeled by a deterministic distribution, denoted by det(k), as the latter describes a constant flow of customers, arriving exactly every k time units (Bertoli et al., 2018).

We thus conduct an experiment aimed at finding convenient IL patterns in terms of *NormalActuate* and *CriticalActuate* response times, i.e. the performance indices of interest. By means of a probability-based CSP, we devise centralized, collaborative and hybrid IL patterns, as reported in Table 2. Performance are assessed with respect to several inter-arrival time (deterministic) distribution means, towards a sensitivity analysis modulo different workload intensities.

The investigation starts with the identification of a workload intensity that can be faced by all the patterns without saturation. We thus tried different mean values for the inter-arrival time distribution, finding 2.5 as a reasonable workload intensity that results into the (heterogeneous) mean response times reported in Table 2[7].

[5]http://www.cplex.com/.

[6]In order to avoid unnecessary complexity in the interpretation of performance analysis results, all the controllers have the same MAPE processing times.

[7]All the QN models have been developed and simulated within the JSimGraph tool available within JMT 1.0.3, running onto a machine equipped with an Intel Core i5 CPU and 16 GB of DDR3 RAM at 1867 MHz. For sake of space, we do not provide simulation configuration parameters. However, they can be found in the JSimGraph .jsimg file at https://gofile.io/?c=S5gWYV, which can be used to replicate the experiment and to devise new ones.

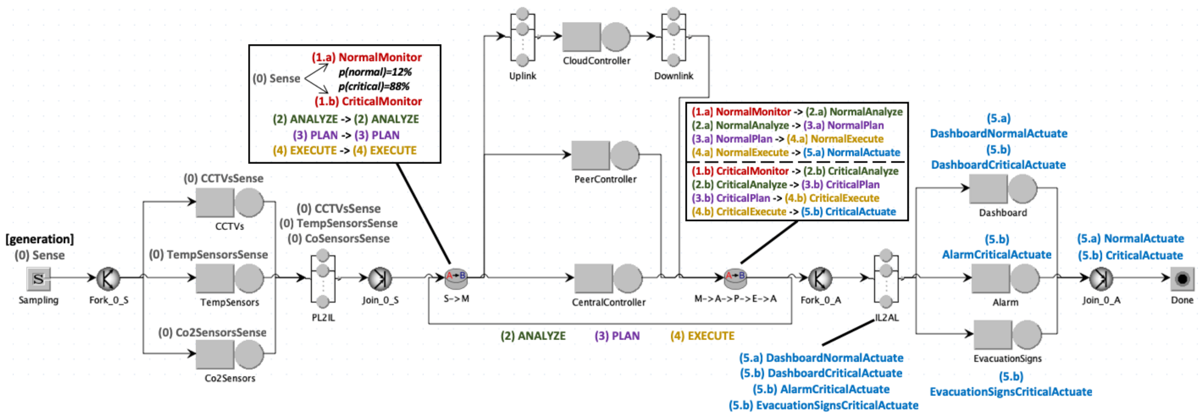

(1.a) NormalMonitor
(0) Sense
p(normal)=12%
p(critical)=88%
(1.b) CriticalMonitor
(2) ANALYZE -> (2) ANALYZE
(3) PLAN -> (3) PLAN
(4) EXECUTE -> (4) EXECUTE
Uplink
CloudController
Downlink
PeerController
(1.a) NormalMonitor -> (2.a) NormalAnalyze
(2.a) NormalAnalyze -> (3.a) NormalPlan
(3.a) NormalPlan -> (4.a) NormalExecute
(4.a) NormalExecute -> (5.a) NormalActuate
(1.b) CriticalMonitor -> (2.b) CriticalAnalyze
(2.b) CriticalAnalyze -> (3.b) CriticalPlan
(3.b) CriticalPlan -> (4.b) CriticalExecute
(4.b) CriticalExecute -> (5.b) CriticalActuate
(5.a) DashboardNormalActuate
(5.b) DashboardCriticalActuate
Dashboard
(0) CCTVsSense
CCTVs
(0) CCTVsSense
(0) TempSensorsSense
(0) CoSensorsSense
[generation]
(0) Sense
Sampling
Fork_0_S
(0) TempSensorsSense
TempSensors
PL2IL
Join_0_S
S->M
CentralController
M->A->P->E->A
Fork_0_A
IL2AL
(5.b) AlarmCriticalActuate
Alarm
(5.a) NormalActuate
(5.b) CriticalActuate
Join_0_A
Done
(0) Co2SensorsSense
Co2Sensors
(2) ANALYZE
(3) PLAN
(4) EXECUTE
(5.a) DashboardNormalActuate
(5.b) DashboardCriticalActuate
(5.b) AlarmCriticalActuate
(5.b) EvacuationSignsCriticalActuate
EvacuationSigns
(5.b) EvacuationSignsCriticalActuate

Figure 4: SMAPEA QN for the considered case study. For sake of illustration $S \rightarrow M$ routing probabilities are not reported, due to the fact that they vary during the experimentation.

Table 2: Considered patterns and corresponding $S \rightarrow M$ routing probabilities (*rp*). Performance indices of interest, i.e. *NormalActuate* and *CriticalActuate* mean response times under det(2.5) workload, are also reported.

Pattern		Normal/Critical				System Response Time (s)	
Name	**Routing probabilities**	**M**	**A**	**P**	**E**	**NormalActuate**	**CriticalActuate**
`Centralized.local`	rp(CentralController)	1	1	1	1	0.4828	3.6604
`Centralized.remote`	rp(CloudController)	1	1	1	1	5.4192	8.7421

`Collaborative`	rp(CentralController)
	rp(PeerController)
		0.5
		0.5
			0.5
			0.5
				0.5
				0.5
					0.5
					0.5
						0.4313	3.6488

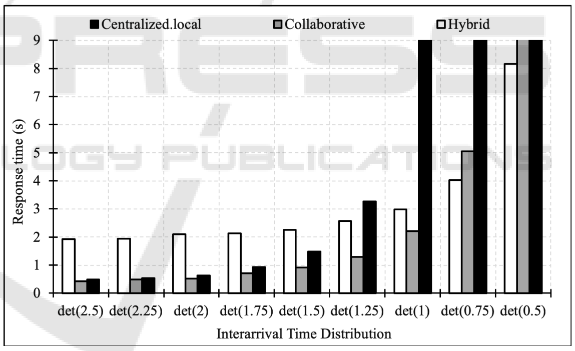
`Hybrid`	rp(CentralController)
	rp(PeerController)
	rp(CloudController)
		0.334
		0.333
		0.333
			0.334
			0.333
			0.333
				0.334
				0.333
				0.333
					0.334
					0.333
					0.333
						1.9206	5.1792

We observe that `Centralized.remote` performs much worse than the other patterns. This is intuitive, because in such pattern any job spends positive delays at *Uplink* and *Downlink* in order to be processed by *CloudController*, which has the same demands of local controllers. These delays make `Centralized.remote` "dominated" by `Centralized.local` by construction[8], hence the former can be immediately discarded.

The investigation continues by considering more intense workloads. The mean value of the deterministic inter-arrival time distribution is thus decreased by 0.5 (seconds) at each step of the sensitivity analysis. Simulation results under the 9 workload intensities are shown in Figure 5. Response times increase exponentially, as expected, since QN stations service demands are modeled as exponential distributions and workloads as deterministic. For the last three workload intensities (i.e. 1, 0.75 and 0.5, respectively), at least one pattern saturates, hence those workloads are referred as "most intense" (conversely, the remaining six workloads are referred as "least intense").

The obtained results pave the way to several hints:

- `Collaborative` manages least intense workloads better than the other patterns. Also, it can manage



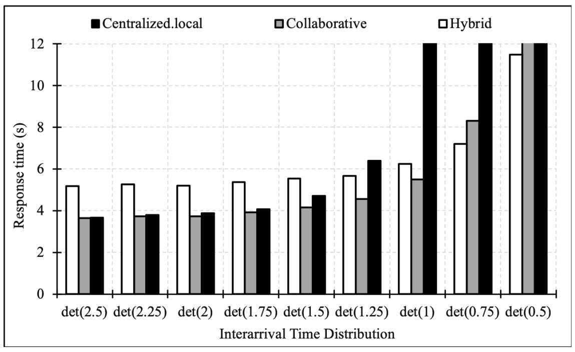


Figure 5: Patterns mean response times for *NormalActuate* (top) and *CriticalActuate* (bottom) under different workload intensities. Values overcoming the maximum y value denote saturation.

[8]A pattern *dominates* another pattern when the former shows both *NormalActuate* and *CriticalActuate* mean response times lower than the latter.

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

two out of the three most intense workloads, while dominating `Hybrid` once (i.e. for det(1)).

- `Centralized.local` can manage least intense workloads only, for which it represents the second better option, except for det(1.25), which can be better handled by `Hybrid` (and, optimally, by `Collaborative`).
- `Hybrid` is the only pattern able to handle all the considered workloads. However, it represents the best solution for the two most intense workloads only, while being dominated by (i) `Collaborative` for det(1) and det(1.25) and (ii) any other pattern for the remaining least intense workloads.

Consequently, architectural decisions could be carried out based on specific requirements, e.g.:

- If saturation is not allowed, then `Hybrid` is the unique option. However, it is dominated by both the other patterns under five out of six least intense workloads, while worsening its performance by few seconds for *NormalActuate* and fractions of second for *CriticalActuate*; This could represent an acceptable trade-off in a real context, as degradation mostly affects the non-critical scenario (i.e. *NormalActuate)*.
- If saturation is allowed, then `Collaborative` may represent the best solution, since it dominates any other pattern for seven out of the nine considered workloads (i.e. $77.\overline{7}\%$). Moreover, under the remaining 2 workloads (i.e. det(0.75) and det(0.5)), it is dominated by `Hybrid` only.
- As a consequence of the two points above, `Centralized.local` is always dominated by at least one of the other patterns hence, in general terms, it would never be chosen. However, economic constraints could inhibit some patterns, thus forcing the consideration of non-optimal solutions like that.

5 CONCLUSION

In this paper, we have introduced a generalized QN model enabling performance modeling and assessment of SASSs, aimed at supporting performance-driven architectural decision-making. Differently from existing approaches, our model includes sensing and actuating components, thus widening the scope of performance modeling and assessment to the whole system. Jobs flowing through the QN represent sequences of Sense, Monitor, Analyze, Plan, Execute and Actuate (`SMAPEA`) activities, each visiting specific parts of the model. A global `MAPE` control loop is implemented by two class-switches embracing system's controllers and operating in conformance to predefined class-switching and routing probabilities: the former are used to progress through the control loop, whilst the latter implement controller selection policies.

We have preliminary evaluated our work with respect to a realistic case study for emergency response, by a workload-driven sensitivity analysis of four architectural patterns. Results have shown that the generalized QN model can usefully support decision-making towards convenient alternative architectures.

However, further experiments are needed in order to enforce the validity of this work. Such experiments might represent an opportunity for investigating additional controller selection policies deriving from the different routing strategies available in JMT, especially the ones considering controllers' metrics at the time the routing is performed (e.g. routing to the controller showing the shortest queue length), which seem particularly suitable to introduce dynamic adaptation of the controller selection policy.

The generalized QN model introduced in this paper supports a single `MAPE` loop. This does not allow to model decentralized architectural patterns such as the ones introduced by Weyns et al (Weyns et al., 2013), where different `MAPE` loops (or parts of them) interact each other and participate to the adaptations concerning other portions of the system. To this aim, we plan to address `SMAPEA` QNs "composition", i.e. two ILs belonging to different QNs might be connected such that controllers within an IL can process activities from the other IL.

In terms of practical applicability, we plan to develop a pattern-based model-driven framework aimed at introducing automation in performance modeling, analysis and decision-making, along SASS's lifecycle. Key-capabilities of such framework might be: (i) designing `SMAPEA` QNs; (ii) generating huge sets of `SMAPEA` QNs, analyzing them and suggesting convenient alternative architectures; (iii) deriving `SMAPEA` QNs from existing SASSs and/or from their architecture models conforming to modeling notations that allow to embed performance parameters (e.g. workload, demands, indices), such as UML (Rumbaugh et al., 2004) with MARTE (Selic and Grard, 2013).

ACKNOWLEDGEMENTS

This research has been funded by the Italian Ministry of Education, University and Research – MIUR, L. 297, art. 10.

REFERENCES

Arbib, C., Arcelli, D., Dugdale, J., Moghaddam, M. T., and Muccini, H. (2019). Real-time Emergency Response through Performant IoT Architectures. In *International Conference on Information Systems for Crisis Response and Management (ISCRAM)*.

Arcelli, D. and Cortellessa, V. (2013). Software model refactoring based on performance analysis: better working on software or performance side? In Buhnova, B., Happe, L., and Kofron, J., editors, *FESCA*, volume 108 of *EPTCS*, pages 33–47.

Arcelli, D., Cortellessa, V., Filieri, A., and Leva, A. (2015). Control theory for model-based performance-driven software adaptation. In *QoSA*, pages 11–20. ACM.

Arcelli, D., Cortellessa, V., and Leva, A. (2016). A library of modeling components for adaptive queuing networks. In *EPEW*, volume 9951 of *LNCS*, pages 204–219. Springer.

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., and Zhang, S. (2019). Proteus: Language and runtime support for self-adaptive software development. *IEEE Software*, 36:73–82.

Becker, M., Becker, S., and Meyer, J. (2013). Simulizar: Design-time modeling and performance analysis of self-adaptive systems. In Kowalewski, S. and Rumpe, B., editors, *Software Engineering*, volume 213 of *LNI*, pages 71–84. GI.

Becker, M., Luckey, M., and Becker, S. (2012). Model-driven performance engineering of self-adaptive systems: A survey. In *QoSA*, pages 117–122. ACM.

Bertoli, M., Casale, G., and Serazzi, G. (2018). Java Modelling Tools – user manual. http://jmt.sourceforge.net/Papers/JMT_users_Manual.pdf. [Online; accessed 20th January, 2020].

Calinescu, R., Grunske, L., Kwiatkowska, M., Mirandola, R., and Tamburrelli, G. (2011). Dynamic qos management and optimization in service-based systems. *IEEE Trans. on Softw. Eng.*, 37(3):387–409.

Cámara, J., Garlan, D., Kang, W. G., Peng, W., and Schmerl, B. R. (2017). Uncertainty in self-adaptive systems categories, management, and perspectives. Technical report, Institute for Software Research, Carnegie Mellon University.

Elkhodary, A., Esfahani, N., and Malek, S. (2010). Fusion: A framework for engineering self-tuning self-adaptive software systems. In *FSE*, pages 7–16. ACM.

Epifani, I., Ghezzi, C., Mirandola, R., and Tamburrelli, G. (2009). Model evolution by run-time parameter adaptation. In *ICSE*, pages 111–121. IEEE Computer Society.

Grassi, V., Mirandola, R., and Randazzo, E. (2009). Model-driven assessment of qos-aware self-adaptation. In Cheng, B. H. C., de Lemos, R., Giese, H., Inverardi, P., and Magee, J., editors, *Software Engineering for Self-Adaptive Systems*, pages 201–222. Springer Berlin Heidelberg.

Incerto, E., Tribastone, M., and Trubiani, C. (2017). Software performance self-adaptation through efficient model predictive control. In *ASE*, pages 485–496.

Jung, G., Joshi, K. R., Hiltunen, M. A., Schlichting, R. D., and Pu, C. (2008). Generating adaptation policies for multi-tier applications in consolidated server environments. In *ICAC*, pages 23–32.

Kephart, J. O. and Chess, D. M. (2003). The vision of autonomic computing. *Computer*, 36(1):41–50.

Kounev, S., Brosig, F., Huber, N., and Reussner, R. (2010). Towards self-aware performance and resource management in modern service-oriented systems. In *ICSC*, pages 621–624.

Lazowska, E. D., Zahorjan, J., Graham, G. S., and Sevcik, K. C. (1984). *Quantitative system performance - computer system analysis using queueing network models*. Prentice Hall.

Lung, C., Zhang, X., and Rajeswaran, P. (2016). Improving software performance and reliability in a distributed and concurrent environment with an architecture-based self-adaptive framework. *JSS*, 121:311–328.

Morin, B., Barais, O., Nain, G., and Jezequel, J.-M. (2009). Taming dynamically adaptive systems using models and aspects. In *ICSE*, pages 122–132. IEEE Computer Society.

Muccini, H. and Sharaf, M. (2017). Caps: Architecture description of situational aware cyber physical systems. In *Software Architecture (ICSA), 2017 IEEE International Conference on*, pages 211–220. IEEE.

Musa, J. D. (1993). Operational profiles in software-reliability engineering. *IEEE Software*, 10(2):14–32.

Perez-Palacin, D. and Mirandola, R. (2014). Uncertainties in the modeling of self-adaptive systems: A taxonomy and an example of availability evaluation. In *ICPE*, pages 3–14. ACM.

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

Selic, B. and Grard, S. (2013). *Modeling and Analysis of Real-Time and Embedded Systems with UML and MARTE: Developing Cyber-Physical Systems*. Morgan Kaufmann Publishers Inc., 1st edition.

Shevtsov, S., Berekmeri, M., Weyns, D., and Maggio, M. (2018). Control-theoretical software adaptation: A systematic literature review. *IEEE Trans. on Softw. Eng.*, 44(8):784–810.

Weyns, D., Iftikhar, M. U., de la Iglesia, D. G., and Ahmad, T. (2012). A survey of formal methods in self-adaptive systems. In *C3S2E*, pages 67–79. ACM.

Weyns, D., Schmerl, B., Grassi, V., Malek, S., Mirandola, R., Prehofer, C., Wuttke, J., Andersson, J., Giese, H., and Göschka, K. M. (2013). On patterns for decentralized control in self-adaptive systems. In *Software Engineering for Self-Adaptive Systems II*, volume 7475 of *LNCS*, pages 76–107. Springer, Berlin, Heidelberg.

Zhang, X., Lung, C., and Franks, G. (2010). Towards architecture-based autonomic software performance engineering. In Drira, K., editor, *CAL*, volume L-5 of *Revue des Nouvelles Technologies de l'Information*, pages 144–156. Cépaduès-Éditions.