Modular Quality-of-Service Analysis of Software Design Models for Cyber-Physical Systems

Riccardo Pinciroli[1][0000−0003−3375−7256], Raffaela Mirandola[2][0000−0003−3154−2438], and Catia Trubiani[1][0000−0002−7675−6942]

[1] Gran Sasso Science Institute, Italy
{riccardo.pinciroli,catia.trubiani}@gssi.it
[2] Politecnico di Milano, Italy raffaela.mirandola@polimi.it

Abstract. Emerging applications such as collaborative and autonomous cyber-physical systems (CPS) seek for innovative techniques that support Quality-of-Service (QoS) analysis as key concern to be considered. The objective of this paper is to complement the software design models with an approach that provides a set of *modules* that are (i) representative of multiple QoS-based properties, and (ii) equipped with strategies aimed to establish rules of interaction among them in a feedback loop fashion. We propose a novel methodology that builds upon the specification of QoS-based modules and enables the generation of design alternatives as outcome of an internal intertwining of different QoS analysis results for CPS. The approach is applied to a collaborative and autonomous network of sensors, and experimental results show that software designers are supported in the selection of design alternatives by quantitative information. A comparison with an integrated model is performed to show the advantages of our novel modular QoS-based analysis.

Keywords: Software Design Models · Quality-of-Service Analysis · Modularity · Cyber-Physical Systems.

1 Introduction

Cyber-physical systems (CPS) are complex systems where both hardware and software components interact together in a tight way to offer the required services. This complexity exacerbates when considering CPS that are *collaborative* [18] (i.e., components collaborate and establish new services for mutual benefit) and *autonomous* [20] (i.e., components take the initiative of interacting with other components for mutual advantage). These characteristics of CPS may reveal a major impact when evaluating the quality of applications, since quality-of-service (QoS) attributes (e.g., energy consumption and performance) are affected by collaborative and autonomous behaviours of system components.

In the literature, early validation of QoS-based requirements and their continuous monitorability has been assessed as fundamental in the software development process [23], and several methodologies have been defined to analyze services enriched with annotations about their QoS attributes [29]. However,

the problem to combine the behaviour of system components (and their quality) in an appropriate way is still challenging [26], even more so if we consider that each QoS attribute may bring its own specification. Hence, it becomes necessary to integrate multiple formalisms and supporting their heterogeneity.

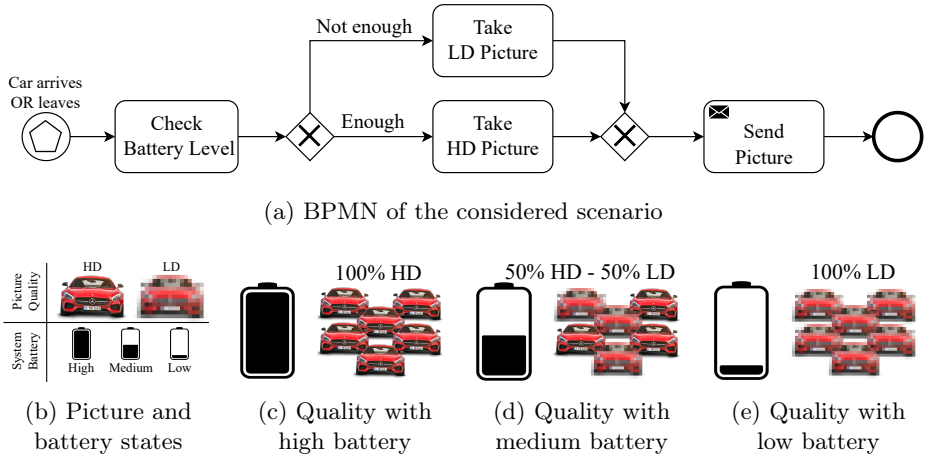
In this paper, we investigate the problem of bridging multiple and heterogeneous QoS attributes to support their collaborative and autonomous analysis, thus reflecting the characteristics of CPS, with the underlying goal of achieving a mutual profit among the attributes. Consider as an example, camera devices that operate in different modes on the basis of their battery levels. Dependencies arise between QoS attributes, e.g., battery charge impacts the quality of the pictures, and consequently the system performance. State-of-the-art approaches *separately* derive the QoS attributes and combine them afterward by analyzing the Pareto front, adopting ad-hoc weighted sums, or studying their trade-off [7,16,25]. Instead, we pursue an approach that aims to establish rules of *interaction* among components and their QoS analysis results. This means to decide, in a feedback loop fashion, when a QoS-based model (due to its analysis results) is supposed to trigger design changes required by another QoS-based specification.

To evaluate the quality of collaborative and autonomous CPS we need a change of perspective w.r.t. more traditional systems. We claim the need for a modular multi-view approach, namely `MODULO` (MODular qUaLity-of-service mOdels), that adopts different connected models to help the software designer in selecting alternatives for collaborative and autonomous CPS. The main advantages of this approach are as follows: (i) it puts together different aspects of the system still maintaining a separation of concerns; (ii) it avoids the complexity of defining a unique flat model including all the system aspects; (iii) the adoption of QoS-based *modules* (i.e., models of a specific QoS characteristics of the systems) allows the plug-and-play of different models that focus on the attribute of interest, or that reflect the different knowledge about the system itself.

Our approach allows the continuous interaction of collaborative and autonomous QoS-based modules, where the results of a module can feed a different one in a feedback loop fashion to empower the system quality evaluation. Similarly to all model-based analysis approaches, numerical values of input parameters can be customized by software designers (reflecting end-users expectation) to grasp different operational profiles and/or varying software and hardware characteristics. The main advantage of the proposed approach is to provide quantitative information to software designers that are supported in the selection of design alternatives. This is achieved through a novel modular analysis technique that considers the intertwining of different QoS-based modules interacting on the basis of model-based analysis results.

The rest of the paper is organized as follows. Section 2 motivates our investigation through an illustrative scenario. Section 3 describes our approach, and Section 4 reports the experimental results of a case study[3]. Section 5 discusses the main limitations of the approach. Section 6 presents related work, and Section 7 concludes the paper by outlining future research directions.

[3] Replication package: https://doi.org/10.5281/zenodo.7773975



(a) BPMN of the considered scenario

(b) Picture and battery states

(c) Quality with high battery

(d) Quality with medium battery

(e) Quality with low battery

Fig. 1: The analyzed application: a collaborative and autonomous smart parking system where the camera resolution depends on the battery charge.

2 Motivating Example

Inspired by [4], we consider the case of a collaborative and autonomous smart parking as a motivating scenario. We assume that, in the analyzed system, there are battery-powered cameras that observe cars entering/leaving a parking lot. Fig. 1(a) depicts the BPMN of the picture collection process.

When a car enters or leaves the parking lot, the system checks the camera battery level. Cameras take HD or LD pictures of the car depending on their battery level. Collected pictures are forwarded to a cloud application where they are analyzed, e.g., to charge car owners based on the length of their staying in the parking lot. Fig. 1(b) depicts camera working modes, i.e., low and high definition modes (LD and HD, respectively). In the former case, cameras capture LD pictures, while high-quality pictures (i.e., with more details) are taken in the latter one. For example, when the cloud application analyzes LD pictures, it may be able to extract only a few information, such as car color and body style (e.g., hatchback, sedan, wagon). The cloud application may get more data from HD pictures, e.g., the registration plate and the car manufacturer.

When a new picture is captured, some energy is consumed and the battery state-of-charge (SoC) decreases; taking a LD picture uses less energy than collecting a HD one. The system battery is characterized by different states defined by its available charge. For the sake of the presentation, in this section, we consider only three battery states (i.e., high, medium, and low charge); more states are considered when `MODULO` is evaluated in Section 4. Cameras autonomously adapt their working mode to the battery SoC to make the battery live longer, i.e., delay the discharge process. For example, when the battery is in the *high charge* state, all cameras work in HD mode, see Fig. 1(c). When the battery SoC decreases and less energy is available, some cameras work in HD mode while oth-

ers operate in LD mode as in Fig. 1(d). If the battery is almost discharged, see Fig. 1(e), all cameras capture LD pictures to reduce the required energy.

The smart parking goal is to maximize a score, i.e., low- and high-quality pictures provide a reward ρ_{ld} and ρ_{hd}, with $\rho_{ld} < \rho_{hd}$, depending on the information contained in each picture. If cameras work only in HD mode, they generate highly rewarding pictures for a short time (i.e., the battery drains fast). If they work in LD mode, the battery lasts longer, but collected pictures provide a small reward. Reasons that make this application well-suited for our analysis are discussed in the following: (i) the system is characterized by three main QoS attributes (i.e., battery depletion, system performance, and reward computation) that need to be modeled with their interactions; (ii) the battery depletion and the system performance affect each other in a complex feedback loop fashion; (iii) naive (i.e., static) configurations of the considered system can be modeled by standard approaches, hence allowing comparing `MODULO` to other modeling frameworks.

The smart parking system also emphasizes the strengths of using modular approaches like `MODULO` instead of an integrated one. Specifically, `MODULO` allows modeling systems whose components interact in a complex way, e.g., in the smart parking, we analyze the system reward obtained by continuously adapting the performance of the system to the battery SoC. Moreover, `MODULO` gives software designers the freedom to model each QoS-based module using the preferred solution, e.g., processes of the smart parking system are analyzed using state-of-the-art models and algorithms.

3 Our Approach

The `MODULO` approach aims to empower the software designer with the ability to assess the quality of the system in the early design stage. To this end, quality assurance (QA) experts are involved to define modules and interactions based on the system requirements.

CPS are composed of highly intertwined software and hardware components that interact with the environment to offer their service and achieve their goal. To guarantee the quality objectives of CPS, it is crucial to evaluate how different QoS attributes influence each other. This calls for an interactive modular approach that considers the intertwining of all QoS attributes. To this end, the `MODULO` approach (see Fig. 2) devises the following main operational steps.

Step ① - System Modeling. The first step consists in modeling the software system: services are specified, and key quality attributes A_i (i=0, ..., N, where the value of N depends on the specific software system) are identified. For example, these attributes may be A_1: performance and A_2: energy consumption for a battery powered system, or A_1: reliability (to model possible component malfunctions), A_2: performance, and A_3: security for a health-care system, or A_1: availability (to model the probability of invoking services conditioned by environmental changes) and A_2: performance for a wind-generator system.

Step ② - QoS-based Modules Modeling. A module M_i is associated with each attribute A_i and will include the appropriate model(s) for the evaluation of A_i,

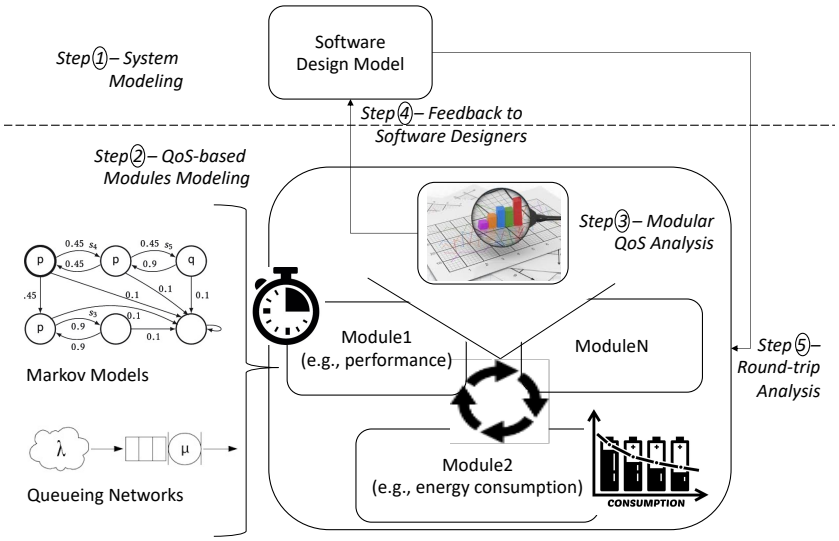


Fig. 2: High-level vision of our MODULO approach.

defined according to the present level of knowledge. For example, if A_i denotes the system performance and we have only a very high level description of the system, a simple Queueing network (QN) model [14] can be defined and evaluated. A deeper knowledge of the system allows the definition of a more detailed and precise QN, Discrete/Continuous-time Markov models [5] (as illustrated in Fig. 2), or any other software performance engineering formalism.

Step ③ - Modular QoS Analysis. Each module M_i is evaluated and results are fed to other modules M_j. Due to the modular nature of the approach, QA experts can adopt different formalisms and analysis techniques for each module. The rules of interaction among modules are meant to define how different aspects of a system affect each other when the behavior depends on the state of the environment or of the system itself [19]. MODULO offers trade-off and parametric sensitivity analysis to identify the system configuration that allows optimizing the satisfaction of quality objectives. This way, software designers are supported with quantitative information on service quality analysis and trade-offs.

Step ④ - Feedback to Software Designers. The software designer is informed about the trade-off and sensitivity analysis of parameters that support their design choices in the early stages of software development. After analysing the received feedback, the QA expert can suggest updates to the software model that represent design alternatives most likely not leading to QoS-based shortcomings. This process enables the early detection and diagnosis of issues, and avoids increased costs to fix errors later in the project life-cycle [12].

Step ⑤ - Round-trip Analysis. The QoS evaluation can be repeated when the knowledge about the whole system or part of it increases. This leads to the definition of more precise models for the corresponding module(s) with subsequent new analysis and evaluation steps. Rules of interaction among modules are not

supposed to change at this stage. Instead, results of the model-based analysis may vary and trigger different QoS attributes changes.

4 Evaluation

In the following, we use the MODULO approach (see Section 3) to analyse the smart parking scenario presented in Section 2. We focus on steps ②–④ which constitute the key novelty of the MODULO. For step ① we plan to adopt state-of-the art methods [6], whereas step ⑤ is left for future work.

4.1 Step ② – QoS-based Modules Modeling

Here, we discuss all modules used to model the three identified QoS attributes (i.e., battery depletion, system performance, and reward computation) of the smart parking. Specifically, each QoS attribute is deployed with different models or analysis techniques. This way, we aim to emphasize the plug-and-play capability of MODULO, as well as its other benefits (i.e., adoption of different formalisms for each module, independent model-based analysis techniques, and interaction among analysis results of different modules). It is worth remarking that the selection of QoS attributes (i.e., battery, performance, and reward) is one of the possible interpretations of the smart parking scenario. The MODULO approach allows implementing other modules and defining further interactions that may differently describe the system and its requirements.

Fig. 3 depicts the overview of MODULO modules developed for analyzing the smart parking. The three modules are drawn as boxes, their interactions are shown using arrows. Interactions are labeled with metrics exchanged between the modules. Specifically, the `Battery Depletion` module passes the number of low- and high-quality requests, i.e., $\vec{N} = (N_{ld}, N_{hd})$, and the number of active cameras C to the `System Performance` module. This module computes the system throughput and response time for the two types of request, i.e., $\vec{X} = (X_{ld}, X_{hd})$ and $\vec{R} = (R_{ld}, R_{hd})$, respectively, as well as the overall sensor utilization, i.e., U. The system throughput and sensor utilization are forwarded to the `Battery Depletion` module that uses these metrics to determine the battery longevity (i.e., L) before reaching the new SoC. This loop is repeated until the battery is fully discharged. $\vec{X}$, $\vec{R}$, and L are continuously communicated to the `Reward Computation` module that uses these values to compute the system reward.

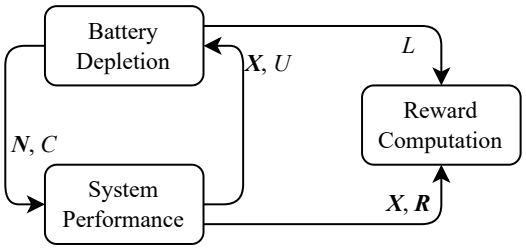


Fig. 3: Modules and their interactions developed using MODULO to study the smart parking system.

Battery Models. We deploy the `Battery Depletion` module using two different models, i.e., the *Linear Model* and the *Kinetic Battery Model* (KiBaM) [17]. These models analytically evaluate the longevity (i.e., L) before moving to the new battery state when the system undergoes a given electrical current load (i.e., I). The battery longevity can be studied using different models, e.g., those discussed in [10]. Our choice of implementing the linear model and the KiBaM aims to show that different existing models and their analysis affect the QoS. Software designers can decide to adopt one of these alternatives based on system requirements. The Linear Model is a naive implementation of the discharge process of a battery, it does not account for two essential battery characteristics (i.e., rate capacity and recovery). For a constant load I, it is defined as:

$$L = \frac{SoC_{init} - SoC_{end}}{I}, \tag{1}$$

where SoC_{init} and SoC_{end} are two battery states, with $SoC_{end} \leq SoC_{init}$. In the case of LD and HD requests, I is computed as:

$$I = \frac{X_{ld} \cdot E_{ld} + X_{hd} \cdot E_{hd} + P_{idle} \cdot (1 - U) \cdot C}{V}, \tag{2}$$

where the energy to capture low- and high-quality pictures is E_{ld} and E_{hd}, respectively, P_{idle} is the power required by cameras to stay on without taking any picture, C is the number of active cameras, and V is the battery voltage. The KiBaM is a simple, non-linear, and effective battery model that represents the battery discharge process using two connected tanks, see Fig. 4. The energy required to serve an external load, i.e., $I(t)$, is taken from the *Available Charge* tank whose capacity at time t is $Q_A(t)$. The *Bound Charge* tank represents energy that is not directly available, but that can flow in the *Available Charge* tank (with rate k) to recover the battery SoC. The capacity of the *Bound Charge* at time t is $Q_B(t)$. The initial available charge ratio is $c = Q_A(0)/[Q_A(0) + Q_B(0)]$. The battery is considered fully discharged when the *Available Charge* tank is empty, however the SoC can recover if some energy is in the *Bound Charge* tank, i.e., $Q_B > 0$. Numerical values used with both battery models are reported in Table 1.

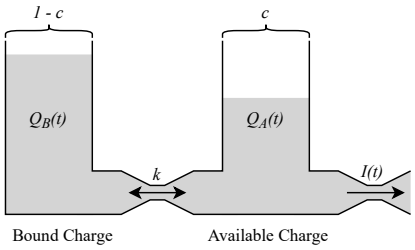


Fig. 4: The Kinetic Battery Model (KiBaM) [17].

Performance Model and Two Analysis Techniques. The smart parking is modeled as a QN to be deployed in the `System Performance` module. Fixed parameters of this module are the think time, $\vec{Z} = (Z_{ld}, Z_{hd})$, the camera service time, $S^{\vec{cam}} = (S^{cam}_{ld}, S^{cam}_{hd})$, and the cloud service times, $S^{\vec{cloud}} = \left(S^{cloud}_{ld}, S^{cloud}_{hd}\right)$, whose numerical values are given in Table 1.

Table 1: Numerical values of models used for the three modules in Fig. 3. Energy consumption of low- and high-quality requests, as well as their service time at cameras are derived from [15] assuming 1 MP and 8 MP resolutions for low- and high-quality pictures, respectively.

Module	Parameter	Value	Parameter	Value
`Battery Depletion`	$Q_A(0)$	23220 Watt-sec	E_{ld}	0.0299 Watt-sec
	$Q_B(0)$ (only KiBaM)	23220 Watt-sec	E_{hd}	0.1286 Watt-sec
	c (only KiBaM)	0.5	P_{idle}	0.2254 Watt
	k (only KiBaM)	2.5 Ampere	V	3.8 Volt
`System Performance`	Z_{ld}	0.1 sec	Z_{hd}	0.1 sec
	S_{ld}^{cam}	0.085 sec	S_{hd}^{cam}	0.4 sec
	S_{ld}^{cloud}	1.0 sec	S_{hd}^{cloud}	2.5 sec
	N	100	–	–
`Reward Computation`	ρ_{ld}	1	ρ_{hd}	5
	SLA (only penalty)	2 sec	–	–

Table 2 reports the numerical value of input parameters that depends on the battery SoC, i.e., the number of requests, $\vec{N} = (N_{ld}, N_{hd})$, and the number of active cameras C. The performance analysis makes use of two widely adopted techniques, i.e., Mean Value Analysis (MVA) and its approximate solution [14]. MVA is an iterative algorithm based on the *Arrival theorem* [2]. It allows retrieving accurate performance metrics (e.g., system response time and throughput) of closed QN with a contained computational cost. Its time and space complexity depend on the number of service centers and the number of customers. A faster and still accurate solution for closed QN is provided by the approximate MVA technique (AMVA) whose complexity depends only on the number of service centers. Note that we model the system performance with QN and solve it using the (approximate) MVA since this is a standard technique in performance engineering [28]. Other formalisms (e.g., Petri nets or Markov chains) can be adopted to analyze different systems. Moreover, if the scenario is too complex to be solved with MVA (e.g., the workload is dynamic [1]), other solution techniques (e.g., simulation) can be used to analyze the system performance at the cost of a longer computation time.

Reward Models. Two reward models (i.e., with and without penalties) are envisioned for the `Reward Computation` module. The model without penalties (namely *W/O*) always assigns the whole reward to the system, independently of the time spent to capture and analyze a picture. This reward model may be used when timeliness is not a crucial system aspect and is computed as:

$$\rho = (X_{ld} \cdot \rho_{ld} + X_{hd} \cdot \rho_{hd}) \cdot L, \tag{3}$$

where X_{ld} and X_{hd} are the average throughput of low- and high-quality requests (i.e., req/sec, from the `System Performance` module), ρ_{ld} and ρ_{hd} are the reward for every low- and high-quality request that is processed, and L is the battery longevity (in seconds, from the `Battery Depletion` module).

The reward model with penalties (namely W/) allows penalizing slow systems. To this end, a Service Level Agreement (i.e., *SLA*) is provided and the reward is computed as:

$$\rho = (X_{ld} \cdot \rho_{ld} + X_{hd} \cdot \rho_{hd}) \cdot L \cdot \left(1 - \frac{\max(\bar{R} - SLA, 0)}{SLA}\right), \quad (4)$$

where $\bar{R}$ is the average system response time of low- and high-quality requests. Similar to the Battery and Performance models, also in this case, other models that better represent system requirements can be implemented and used. Our choice of modeling the reward as in Eqs. (3) and (4) intends to stress the plug-and-play features of the `MODULO` approach. Numerical values of reward model parameters are reported in Table 1.

Table 2: System configurations with low- and high-quality requests (N_{ld}, N_{hd}), and the number of active cameras C depends on the battery SoC. Cameras are turned on/off so that $96\% \leq U \leq 98\%$.

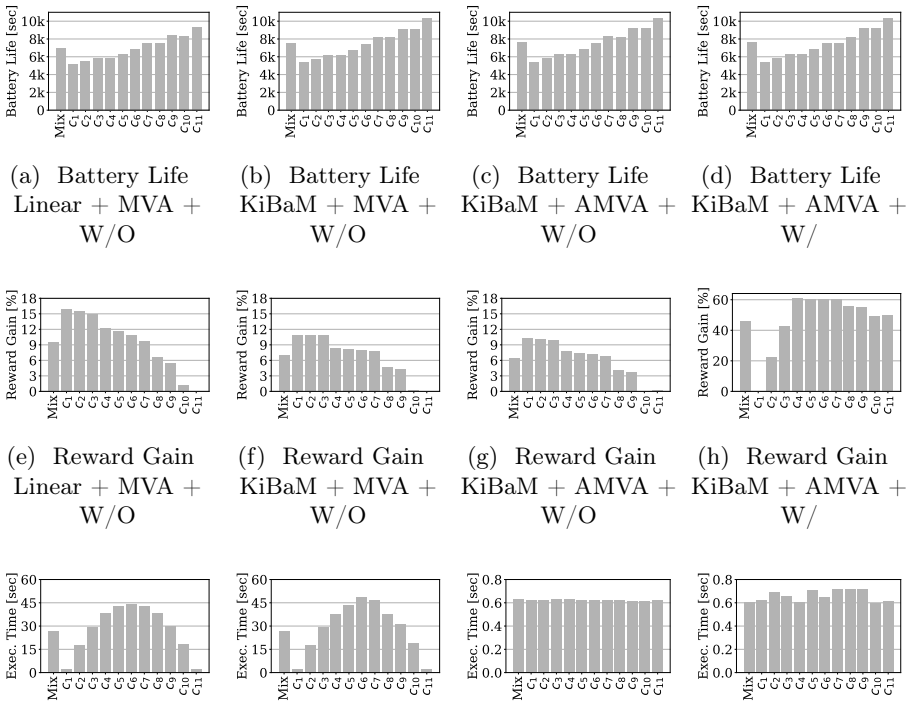
Conf.	N_{ld}	N_{hd}	C	SoC_{init}	SoC_{end}
c_1	0	100	14	100%	90%
c_2	10	90	13	90%	80%
c_3	20	80	12	80%	70%
c_4	30	70	12	70%	60%
c_5	40	60	11	60%	55%
c_6	50	50	10	55%	50%
c_7	60	40	9	50%	40%
c_8	70	30	9	40%	30%
c_9	80	20	8	30%	20%
c_{10}	90	10	8	20%	10%
c_{11}	100	0	7	10%	0%

4.2 Steps ③ and ④ – Modular QoS Analysis and Feedback to Software Designers

Results obtained by analyzing the smart parking using `MODULO` are shown in Fig. 5. On the x-axis the system configuration identifier is reported, see first column of Table 2. Note that the *Mix* configuration means varying the numerical values of parameters (N_{ld}, N_{hd}, and C) during the analysis based on the battery SoC. All other configurations ($c_1, \ldots, c_{11}$) are fixed ones (i.e., N_{ld}, N_{hd}, and C are set at the beginning of the analysis and do not change). Every row of Fig. 5 depicts a different metric: (i) battery depletion time, (ii) reward percentage increment over the configuration with the minimum reward, and (iii) time required by `MODULO` to complete the analysis. Columns account for the effect of different models and analysis techniques, e.g., the first column shows results obtained using the Linear model for the `Battery Depletion` module, the MVA algorithm for the `System Performance` module, and the reward without penalties for the `Reward Computation` module.

The battery longevity is depicted in Figs. 5(a)–(d). The largest variation is observed when the `Battery Depletion` module uses the Linear model, Fig. 5(a), or the KiBaM, Figs. 5(b)–(d). With the Linear model, the battery depletion time is 4 to 15 minutes shorter than the one observed with the KiBaM due to the recovery process modeled only by the latter approach. Models used to deploy the `System Performance` and `Reward Computation` modules do not affect significantly the battery life, i.e., the observed variation is less than 2 minutes.

Figs. 5(e)–(h) show the reward increment obtained by each configuration over the minimum reward, i.e., *Reward Gain [%]* $= [(\rho - \rho_{min})/\rho_{min}] \cdot 100$, where ρ is



(a) Battery Life Linear + MVA + W/O (b) Battery Life KiBaM + MVA + W/O (c) Battery Life KiBaM + AMVA + W/O (d) Battery Life KiBaM + AMVA + W/

(e) Reward Gain Linear + MVA + W/O (f) Reward Gain KiBaM + MVA + W/O (g) Reward Gain KiBaM + AMVA + W/O (h) Reward Gain KiBaM + AMVA + W/

(i) Execution Time Linear + MVA + W/O (j) Execution Time KiBaM + MVA + W/O (k) Execution Time KiBaM + AMVA + W/O (l) Execution Time KiBaM + AMVA + W/

Fig. 5: System behavior (i.e., battery life, reward gain), and scalability (i.e., execution time) using `MODULO` with different models and analysis techniques.

the reward of the considered configuration and ρ_{min} is the minimum reward. The configuration whose *Reward Gain* is 0% scored $\rho = \rho_{min}$. This is the metric that depends the most on the deployed modules since some variations are observed every time a model is changed. When penalties are not included in the reward, i.e., Figs. 5(e)–(g), serving a large number of high-quality requests (i.e., c_1, c_2, c_3) allows maximizing the reward. The smaller gain observed when modeling the battery with KiBaM is due to the longer battery life of configurations with small N_{hd} (i.e., those with small rewards). Since the battery takes longer to run out of energy, there is more time to process requests and increase the minimum reward. The model used for the `System Performance` module slightly affects the system reward, smaller rewards observed with the Approximate MVA are due to the reduced accuracy w.r.t. the exact MVA. When the reward accounts also for penalties, see Fig. 5(h) whose y-axis goes from 0% to 60%, there are new configurations with the maximum gain (i.e., $c_4,\ c_5,\ c_6,\ c_7$). This is due to the long response time of configurations serving mainly high-quality requests.

Figs. 5(i)–(l) depict the time MODULO takes to analyze the smart parking with considered configurations. The scalability of MODULO is highly affected by the technique used to analyze the System Performance (i.e., MVA or Approximate MVA). This is due to the different time complexity of the two algorithms, see Section 4.1. Despite system results (i.e., battery life and reward gain) obtained with the exact and approximate MVA are very similar, see Figs. 5(b)–(c) and 5(f)–(g), the approximate solution allows MODULO to complete its analysis in less than a second, note the different y-axis scale.

4.3 Validation of MODULO

To evaluate the accuracy of MODULO, we analyze the smart parking using an *integrated* (i.e., non-modular) model. Specifically, we use a Queueing Petri Nets (QPN) [13] due to its capability of modeling performance and reliability aspects. The choice of using QPN as a benchmark method to compare MODULO is motivated by our interest in highlighting the modular analysis as a key contribution, and QPN nicely fits with the need of showing differences w.r.t. an integrated model. Other related works (see Section 6) are excluded from comparison, since they separately evaluate quality attributes and eventually combine them using techniques such as Pareto analysis.

Fig. 6 depicts the QPN model used in this section for comparison, where white (gray) transitions are driven by Exponential (Deterministic) processes, and solid (dashed) arrows describe the performance (battery) workflow. Note that QPN does not support the online adaptation of system configuration to the observed battery SoC, only fixed configurations are analyzed. Moreover, for the sake of simplicity, the developed QPN assumes a linear battery consumption. *These are two advantages of using MODULO, i.e., it allows analyzing collaborative and autonomous systems, and enables an ease adoption of different models.*

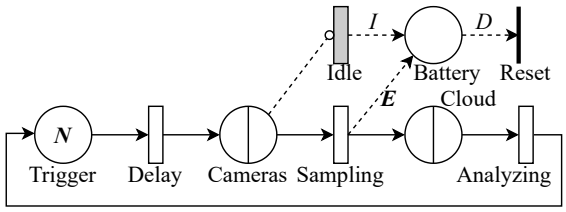
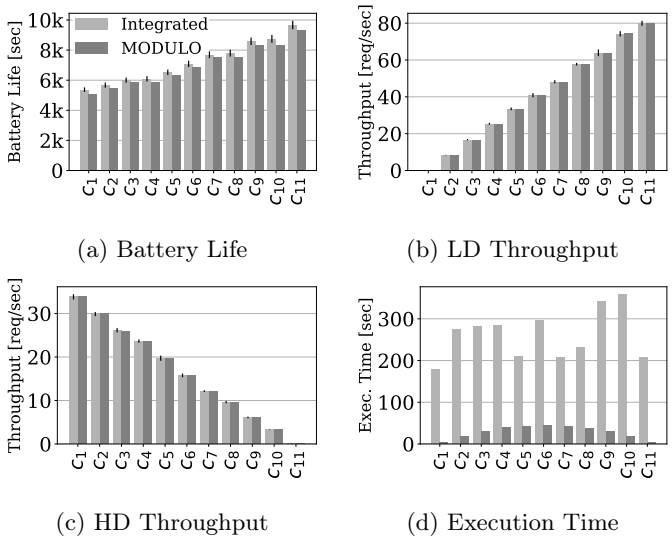


Fig. 6: QPN model of the smart parking.

For a fair comparison, MODULO is deployed with the Linear battery model and the MVA algorithm. The performance aspect is modeled as follows: $\vec{N} = (N_{ld}, N_{hd})$ requests are in the Trigger place that, after an exponentially distributed Delay (with average Z_{ld} and Z_{hd}, see Table 1), move to the Cameras place. Here, requests are processed, low- and high-quality pictures are taken (i.e., Sampling transition) and sent to the Cloud application to be analyzed (i.e., Analyzing transition), before returning to the initial place. The system throughput is measured observing the throughput of low- and high-quality requests at the Analyzing transition. To model the battery consumption aspect in the proposed QPN, every time a picture is captured, $\vec{E} = (E_{ld}, E_{hd})$ tokens are inserted into the Battery place depending on the quality of the taken picture. Energy

is consumed also when cameras are idle (i.e., `Cameras` is empty), in this case I tokens are generated into the `Battery` place every $S_{Idle} = 1$ second. When D tokens are collected into the `Battery` place, the immediate transition `Reset` fires. In this case, the average battery longevity is $L = 1/\phi_{Reset}$, where ϕ_{Reset} is the firing frequency of the `Reset` transition.

Comparison of the two approaches, i.e., `MODULO` and Integrated (in this case a QPN) is shown in Fig. 7, where we consider prediction (i.e., battery life, low-, and high-quality throughput) and scalability perspectives. Battery life, low-, and high-quality throughput are the metrics of choice since they are used (together with the response time that is derived from the response time law [14] through N, X, and Z) to compute the system reward, Eqs. (3) or (4). Since `MODULO` uses analytical formulas to model the smart parking system, exact values are obtained by solving equations presented in Section 4.1. The QPN is simulated on a commodity machine (Intel Core i7-10750H CPU @ 2.60GHz and 16GB memory) using the discrete-event simulator of the Java Modelling Tools [3], a widely used tool for performance analysis. Average values are collected with 99% confidence interval. Fig. 7(a) depicts the battery life estimated



(a) Battery Life (b) LD Throughput

(c) HD Throughput (d) Execution Time

Fig. 7: `MODULO` vs. QPN integrated approach – *prediction* and *scalability*.

by the two approaches for different configurations. The observed error, i.e., $Error\ [\%] = (|M_{Integrated} - M_{MODULO}|/M_{Integrated}) \cdot 100$, is always smaller than 5%, and `MODULO` underestimates the battery depletion time. The throughput of low- and high-quality requests estimated by the two approaches are very similar and errors are negligible, see Figs. 7(b) and 7(c), respectively. Figure 7(d) shows that the scalability of the two approaches largely differs. `MODULO` takes less than a minute to generate results (and it is even shorter when the approximate MVA is used for the `System Performance` module, see Fig. 5).

4.4 Threats to Validity

Internal validity. Threats are caused by bias in establishing cause-effect relationships in our experiments. To limit these threats, we extensively experiment with the considered system. We vary input parameters, look at different output metrics, and compare MODULO to a widely used mathematical modeling language (i.e., QPN). We also make code of MODULO and replication data publicly available.

External validity. Threats arise due to modeling assumptions. To mitigate these threats and assess the appropriateness of claims in Section 4, we use models and data inspired by established case studies in the literature [4,15]. We are aware that the generalization of results is not guaranteed, the application of MODULO to additional scenarios and other domains is left for future work.

Construct validity. Threats related to the design of experiments. We state the purpose of our experimentation, i.e., (i) assessment of MODULO efficacy in modeling collaborative and autonomous CPS and (ii) validation of MODULO results. We analyze different system configurations and show that MODULO can model collaborative and autonomous CPS (e.g., the smart parking with the *Mix* configuration). We compare MODULO to QPN on performance and energy metrics.

Conclusion validity. Threats are due to incorrect relationships between input parameters and model outputs. Results obtained through MODULO are derived from state-of-the-art equations and algorithms whose accuracy is assessed in the literature [14,17]. Results simulated using the QPN are provided with 99% confidence intervals and a maximum relative error of 3%.

5 Discussion

The choice of models for each module is an open issue in the QoS domain [11], and there is no silver bullet for this scope. Modules can be implemented differently, e.g., the system performance can be evaluated using Markov models or QN. Such choices depend on the application domain and are delegated to QA experts. This is a limitation of MODULO since the required expertise might not be available in the development team. In the future, we want to explore the effect of different models on the accuracy of our approach as well as on module interactions.

MODULO is applied to a case study concerned with battery depletion, system performance, and reward computation. Other scenarios may call for different QoS attributes (e.g., reliability and availability) and models. For instance, the system reliability (i.e., malfunction of components) can be captured by Fault-Tree Analysis [21], whereas environmental changes affecting the system (e.g., the speed of the wind may activate different turbines, or the illumination may trigger more/less powerful sensors) is modeled with Bayesian Networks in [24]. As future work, we foresee an orchestrator component that is in charge of capturing the system run-time changes by (i) sensing the environment, (ii) adding the proper modules, and (iii) triggering QA experts for defining the corresponding models and the interactions with other modules already specified.

When considering many QoS attributes, MODULO complexity inevitably increases due to the large number of modules and the difficulty of defining their

interactions. If non-analytic solutions are needed to study one or more modules, the execution time to solve the overall model may increase as well. In Figures 5(i)–(l), we investigate the impact of different solution techniques on MODULO execution time, relatively to the considered case study. In the future, we plan to investigate the effect of increasing the number of modules and their interactions on the scalability of our approach.

6 Related Work

The work presented in this paper relates to all the approaches concerned with QoS-based analysis of software design models, e.g., [9,29]. Seminal work on non-functional requirements and their correlation through softgoal interdependency graphs is presented in [8]. MODULO extends [8] by enabling the definition of interactions among different modules and their parameters. Here, we discuss recent techniques that are closely related to MODULO, and we group them into those more concerned with: (i) the relationships between quality attributes and design choices, and (ii) the trade-off analysis between multiple QoS properties.

Relationships between quality attributes and design choices. Lytra et al. [16] confirm that quality attributes are the major driver of the design process to achieve high quality systems, and they investigate the relationships between quality attributes and design decisions. They observe that performance is the most frequently analyzed attribute, and the cost is also well perceived as a rewarding factor. Differently from [16] we contribute to improve performance as effort from the analysis of other QoS-based system properties. Verginadis et al. [26] consider applications with alternative architecture variants that can be optimized only considering all their services. They extend two modeling frameworks (i.e., CAMEL and MDS) to improve the analysis of such applications in a multi-cloud environment. Despite sharing similar goals, MODULO profoundly differs from the approach discussed in [26]. Specifically, the approach proposed by Verginadis et al. applies to a single domain (i.e., cloud applications) and allows software developers to use the provided language to study concepts already implemented in the modeling framework. MODULO allows using any modeling language to define different services, study their interactions, and analyze the effect of design decisions on QoS attributes. Schneider et al. [22] consider informal knowledge to improve design decisions in software architectures by annotating design models for qualitative-quantitative reasoning. Differently from [22], MODULO does not explore the whole design space. It gets alternative decisions by analyzing results obtained from the interaction of different QoS-based modules.

Trade-off analysis between multiple QoS properties. Vitali [27] presents the Sustainable Application Design Process (SADP) to support software developers in designing sustainable microservice applications. The proposed methodology allows developers to specify different working modalities that account for the trade-off between power consumption and other QoS attributes (e.g., system performance). While the purpose of the SADP methodology is to draw the developers' attention to the sustainability of their applications, MODULO allows analyzing any set of QoS attributes and their interactions. Vale et al. [25] present

an empirical study on quality trade-offs involving patterns when designing microservices, and performance emerges as a key concern in industry practice, in conjunction with scalability that is also perceived as a major challenge in software development. Our work inherits from [25] the goal of supporting software designers when evaluating different QoS analysis results. Cámara et al. [7] explicitly link design decisions to satisfaction of quality requirements, while facilitating the comprehension of trade-offs. Our work differs from [7] since we do not use machine learning techniques to explore the design space, we generate specific design alternatives out of QoS analysis results.

Summarizing, to the best of our knowledge, approaches in the literature consider quality trade-offs design decisions that are supported by an integrated analysis of system models. The `MODULO` novelty is in the underlying reasoning engine that builds upon a modular analysis of different QoS-based properties. The interaction of those properties driven by model-based results allows identifying the most appropriate design alternative.

7 Conclusion and Future Work

In this paper we present `MODULO`, a modular approach that enables the interaction among different quality-of-service (QoS) properties. Its novelty relies on the interplay of results from the model-based analysis of different modules. Our approach is viable when studying intertwined QoS-based properties, each one with its own model and showing interacting features. Experiments assess the benefits of adopting `MODULO` to analyse the quality of cyber-physical systems. As future work, we plan to (i) investigate the last step of `MODULO` (i.e., Round-trip Analysis) and its integration with other steps; (ii) extend the modeling of other interacting QoS properties since we are interested to consider a larger number of relationships; (iii) apply the approach to industrial applications from different domains, to further investigate its usability and scalability.

Acknowledgements. We thank the anonymous reviewers for their valuable feedback. This work has been partially funded by MUR PRIN project 2017TWRCNB SEDUCE, and the PNRR MUR project VITALITY (ECS00000041) Spoke 2 ASTRA - Advanced Space Technologies and Research Alliance.

References

1. Ali, A., et al.: It's not a Sprint, it's a Marathon: Stretching Multi-resource Burstable Performance in Public Clouds. In: Int. Middleware Conf. pp. 36–42 (2019)
2. Asmussen, S.: Applied probability and queues, Second Edition, Applications of mathematics, vol. 51. Springer (2003)
3. Bertoli, M., et al.: JMT: performance engineering tools for system modeling. Perf. Eval. Review **36**(4), 10–15 (2009)
4. Bock, F., et al.: Smart Parking: Using a Crowd of Taxis to Sense On-Street Parking Space Availability. IEEE Trans. Intell. Transp. Syst. **21**(2), 496–508 (2019)
5. Bolch, G., et al.: Queueing Networks and Markov Chains - Modeling and Performance Evaluation with Computer Science Applications. Wiley (2006)

6. Budgen, D.: Software design. Pearson Education (2003)
7. Cámara, J., et al.: Explaining Architectural Design Tradeoff Spaces: A Machine Learning Approach. In: Eur. Conf. on Softw. Archt. pp. 49–65 (2021)
8. Chung, L., et al.: Non-functional requirements in software engineering, vol. 5. Springer Science & Business Media (2012)
9. Fadda, E., et al.: Optimizing Monitorability of Multi-cloud Applications. In: Int. Conf. of Adv. Inf. Syst. Eng. pp. 411–426 (2016)
10. Gazafrudi, S., Nikdel, M.: Various battery models for various simulation studies and applications. Renewable and Sustainable Energy Reviews **32**, 477–485 (2014)
11. Gerasimou, S., et al.: Synthesis of probabilistic models for quality-of-service software engineering. Autom. Softw. Eng. **25**, 785–831 (2018)
12. Haskins, B., et al.: Error Cost Escalation Through the Project Life Cycle. In: INCOSE Annual Int. Symp. pp. 1723–1737 (2004)
13. Kounev, S., et al.: Introduction to queueing petri nets: modeling formalism, tool support and case studies. In: Int. Conf. on Perf. Eng. pp. 9–18 (2012)
14. Lazowska, E.D., et al.: Quantitative System Performance - Computer System Analysis Using Queueing Network Models. Prentice Hall (1984)
15. LiKamWa, R., et al.: Energy characterization and optimization of image sensing toward continuous mobile vision. In: Int. Conf. on Mob. Syst., Appl., and Serv. pp. 69–82 (2013)
16. Lytra, I., et al.: Quality attributes use in architecture design decision methods: research and practice. Computing **102**(2), 551–572 (2020)
17. Manwell, J.F., McGowan, J.G.: Lead acid battery storage model for hybrid energy systems. Solar energy **50**(5), 399–405 (1993)
18. Nazarenko, A.A., Camarinha-Matos, L.M.: Collaborative Cyber-Physical Systems Design Approach: Smart Home Use Case. In: Adv. Doct. Conf. on Comput., Electr. and Ind. Syst. pp. 92–101 (2021)
19. Pinciroli, R., Trubiani, C.: Performance Analysis of Fault-Tolerant Multi-Agent Coordination Mechanisms. IEEE Trans. on Ind. Informatics (2023), Early Access
20. Platzer, A.: The Logical Path to Autonomous Cyber-Physical Systems. In: Int. Conf. on Quantitative Eval. of Systems. pp. 25–33 (2019)
21. Rao, K.D., et al.: Dynamic fault tree analysis using Monte Carlo simulation in probabilistic safety assessment. Reliab. Eng. Syst. Saf. **94**(4), 872–883 (2009)
22. Schneider, Y., et al.: Using Informal Knowledge for Improving Software Quality Trade-Off Decisions. In: Eur. Conf. on Softw. Archt. pp. 265–283 (2018)
23. Shi, H., et al.: How Big Service and Internet of Services Drive Business Innovation and Transformation. In: Int. Conf. of Adv. Inf. Syst. Eng. pp. 517–532 (2022)
24. Trubiani, C., Mirandola, R.: Continuous rearchitecting of qos models: Collaborative analysis for uncertainty reduction. In: Eur. Conf. on Softw. Archt. pp. 40–48 (2017)
25. Vale, G., et al.: Designing Microservice Systems Using Patterns: An Empirical Study on Quality Trade-Offs. In: Int. Conf. on Softw. Archt. pp. 69–79 (2022)
26. Verginadis, Y., et al.: Data and Cloud Polymorphic Application Modelling in Multi-clouds and Fog Environments. In: Int. Conf. of Adv. Inf. Syst. Eng. pp. 449–464 (2021)
27. Vitali, M.: Towards Greener Applications: Enabling Sustainable-aware Cloud Native Applications Design. In: Int. Conf. of Adv. Inf. Syst. Eng. pp. 93–108 (2022)
28. Woodside, C.M., et al.: The Future of Software Performance Engineering. In: Workshop on the Future of Softw. Eng. (FOSE). pp. 171–187 (2007)
29. Woodside, C.M., et al.: Transformation challenges: from software models to performance models. Software and Systems Modeling **13**(4), 1529–1552 (2014)